SHELDON INSTRUMENTS, INC.
QuVIEW FOR USE WITH LabVIEW SOFTWARE

February 2005

Table of Contents

1.0 QuVIEW Software Installation.
- 1.1 QuVIEW Files Organization
- 1.2 QuVIEW Files Installation.
- 1.3 Adding QuVIEW to LabVIEW Paths.
- 1.4 Viewing QuVIEW Menus Under LabVIEW.

2.0 QuVIEW: DSP resident libraries for Real Time Acquisition, Signal Processing and Control.
- 2.1 Building a QuVIEW Application.
- 2.2 Common Terminals to All QuVIEW VIs.

3.0 Common Queue VIs.
- **3.1 Common Queue: Analog I/O.**
  - 3.1.1 PQ_Analog Input Channel.
  - 3.1.2 PQ_Scaled Analog Input Channel.
  - 3.1.3 PQ_Unscaled Analog Input.
  - 3.1.4 PQ_Clipped Analog Output Channel.
  - 3.1.5 PQ_Clipped Unscaled Analog Output.
  - 3.1.6 PQ_Analog Output Channel.
  - 3.1.7 PQ_Scaled Analog Output Channel.
  - 3.1.8 PQ_Unscaled Analog Output.
  - 3.1.9 PQ_Calibrate.
  - 3.1.10 PQ_Multiple Calibrate.
  - 3.1.11 PQ_Scale.
  - 3.1.12 PQ_PQ Source to IEEE.
  - 3.1.13 PQ Analog I/O: Calibration SubVIs
    - 3.1.13.1 Load Cal File
    - 3.1.13.2 Read Cal Table
    - 3.1.13.3 Read Cal Point
    - 3.1.13.4 Store Cal Table
    - 3.1.13.5 Save Cal File
    - 3.1.13.6 Store Cal Point

- **3.2 Common Queue: Digital I/O.**
  - 3.2.1 PQ_Digital Input Port (24 bit max).
  - 3.2.2 PQ_Aux Digital Input Port (SI-MOD68xx).

3.2.3 PQ_Digital Input Port (Split to 16 bit Pair).
3.2.4 PQ_Port Read (24 bit max).
3.2.5 PQ_Digital Output Port (24 bit max).
3.2.6 PQ_Aux Digital Output Port (SI-MOD68xx).
3.2.7 PQ_Digital Output Port (Join 16 bit Pair).
3.2.8 PQ_Port Write (24 bit max).
3.2.9 PQ_Digital I/O Pin (C3x).

3.3 Common Queue: PQ Counters & Timers

3.3.1 PQ_Counter.
3.3.2 PQ_Event Counter Input (SI-MOD68xx).
3.3.3 PQ_Pulse Input Pin (SI-MOD68xx).
3.3.4 PQ_Pulse Output.
3.3.5 PQ_Pulse Width Computation.
3.3.6 PQ_PWM/Event Counter (CLKx on C3x).
3.3.7. PWM/Event Counter SubVIs (C3x).
3.3.7 Init Serial Port Values.
3.3.8 Init Serial Port Values C3x.
3.3.9 Initialize Serial Port.

3.4 Common Queue: Hardware Devices

3.4.1 Initialize Board.
3.4.2 Configure DSP.
3.4.3 Configure I/O.
3.4.4 SI-DSP SubVIs.
3.4.4.1 Configure C6713-PCI.
3.4.4.2 Configure C6711-PCI.
3.4.4.3 Configure C33-PCI.
3.4.4.4 Configure C31-PCI.
3.4.4.5 Configure CAC.
3.4.4.6 Configure C31-ISA.
3.4.4.7 Configure C30-ISA w/ AppLaunch.
3.4.4.8 Configure C30-ISA.
3.4.4.9 Get Board Info.
3.4.4.10 Reset DSP.
3.4.4.11 Configure PCI Communications (6711 PLX).
3.4.4.12 Configure PCI Communications (C33 PLX).
3.4.4.13 Configure PCI Communications (AMCC).
3.4.4.14 Configure ISA Communications.
3.4.4.15 Configure DLL Communications (x86).
3.4.5 Configure DSP6800 Daughter Module.
3.4.6 Configure DSP6400 Daughter Module.
3.4.7 Configure DSP1600 Daughter Module.
3.4.8 Configure CAC I/O & Launch.

3.5 Common Queue: Queue Utilities

3.5.1 Terminate Queue.

3.5.2 Read from DSP board.
3.5.3 Write to DSP board.
3.5.4 DSP to IEEE.
3.5.5 IEEE to DSP.
3.5.6 Error Chainer.

3.6 Supplemental VIs.

3.6.1 Read PCI NVRAM.
3.6.2 Write PCI NVRAM.
3.6.3 Read PCI Opreg.
3.6.4 Write PCI Opreg.

4.0 Point Queuing Function VIs.

4.1 PQ Constants.

4.1.1 PQ_Constant.
4.1.2 PQ_Latch.

4.2 PQ Arithmetic.

4.2.1 PQ_Add.
4.2.2 PQ_Subtract.
4.2.3 PQ_Multiply.
4.2.4 PQ_Divide.
4.2.5 PQ_Absolute Value.
4.2.6 PQ_Negate.
4.2.7 PQ_Inverse.
4.2.8 PQ_Square Root.
4.2.9 PQ_Inverse Square Root.
4.2.10 PQ_Scale aX+b.
4.2.11 PQ_Polynomial.

4.3 PQ Trig and Log.

4.3.1 PQ_Logarithm Base N.
4.3.2 PQ_Natural Logarithm.
4.3.3 PQ_Logarithm Base 10.
4.3.4 PQ_Power X to Y.
4.3.5 PQ_Exp(e)^(X).
4.3.6 PQ_Exp(10)^(X).
4.3.7 PQ_Sine.
4.3.8 PQ_Cosine.

4.4 PQ Calculus.

4.4.1 PQ_Derivative.
4.4.2 PQ_Integral.
4.4.3 PQ_Slope

4.5 PQ Comparison.

4.5.1 PQ_Equal to Zero?
4.5.2 PQ_Not Equal to Zero?
4.5.3 PQ_Equal?
4.5.4 PQ_Not Equal?
4.5.5 PQ_Less Than?
4.5.6 PQ_Less Than or Equal?
4.5.7 PQ_Greater Than?
4.5.8 PQ_Greater Than or Equal?
4.5.9 PQ_In Range?
4.5.10 PQ_Clip Within.
4.5.11 PQ_Max & Min.
4.5.12 PQ_Trigger.
4.5.12.1 Trigger Conditions.

4.6 PQ Display & Log to Disk.
4.6.1 PQ_Scope.
4.6.2 PQ_Packed Scope.
4.6.3 PQ_WaveScope.
4.6.4 PQ_Synch Scope.vi
4.6.5 PQ_Synch Packed Scope.vi
4.6.6 Common Synchronous Scope Parameters.

4.7 PQ Boolean.
4.7.1 PQ_Logical AND.
4.7.2 PQ_Logical OR.
4.7.3 PQ_Logical XOR.
4.7.4 PQ_Logical NOT.
4.7.5 PQ_Logical NAND.
4.7.6 PQ_Logical NOR.
4.7.7 PQ_Logical NOT XOR.
4.7.8 PQ_JK Flip Flop.
4.7.9 PQ_Bitwise Rotate (24 bits max).
4.7.10 PQ_Bitwise Extraction (24 bits max)
4.7.11 PQ_Bitwise AND (24 bits max).
4.7.12 PQ_Bitwise AND.
4.7.13 PQ_Bitwise OR.
4.7.14 PQ_Bitwise XOR.
4.7.15 PQ_Bitwise NOT.
4.7.16 PQ_Bitwise NAND.
4.7.17 PQ_Bitwise NOR.
4.7.18 PQ_Bitwise NOT XOR.

4.8 PQ Digital Filters.
4.8.1 PQ_IIR Filter.
4.8.1.1 IIR Direct Filter Coefficients Design.
4.8.2 PQ_IIR Cascade Filter.
4.8.2.1 IIR Cascade Filter Coefficients Design.

4.8.3 PQ_FIR Filter.
4.8.3.1 FIR Filter Coefficients Design.
4.8.4 PQ_Shock Response.
4.8.4.1 Smallwood Coefficients.
4.8.5 PQ_LMS_Adaptive Filter.

4.9 PQ Process Control.
4.9.1 PQ_PID with Integral Reset.
4.9.1.1 PID Design.
4.9.2 PQ_Setpoint Ramp Generator.

4.10 PQ Special Functions.
4.10.1 PQ_Average.
4.10.2 PQ_Average Float-Point Compensated.
4.10.3 PQ_Squelch/Noise Gate

4.11 Waveform Generation.
4.11.1 PQ_PlayBack.
4.11.2 Waveform Design.
4.11.3 PQ_Waveform Playback.

4.12 Z Transforms.
4.12.1 PQ_Z Delay.
4.12.2 PQ_Feedback Node.
4.12.3 PQ_Summing Node.
4.12.4 PQ_Multiplying Node.
4.12.5 PQ_Feedback Delay.

4.13 PQ Miscellaneous.
4.13.1 PQ_Add PQ Function to List
4.13.2 PQ_User Function.

5.0 Vector Queuing VIs.

5.1 VQ Constants.
5.1.1 VQ_2D Constants.
5.1.2 VQ_Constant.

5.2 VQ Arithmetic.
5.2.1 VQ_Addition.
5.2.2 VQ_Subtraction.
5.2.3 VQ_Multiplication.
5.2.4 VQ_Division.
5.2.5 VQ_Quotient & Remainder.
5.2.6 VQ_Complex Addition.
5.2.7 VQ_Complex Subtraction.
5.2.8 VQ_Complex Multiplication.

5.2.9 VQ_Complex Division.
5.2.10 VQ_Vector Summation.
5.2.11 VQ_Recursive Vector Summation.
5.2.12 VQ_Summation of Vector Elements.
5.2.13 VQ_Product of Vector Elements.
5.2.14 VQ_Absolute Value.
5.2.15 VQ_Negate.
5.2.16 VQ_Invert.
5.2.17 VQ_Square.
5.2.18 VQ_Square Root.

5.3 VQ Trig and Log.

5.3.1 VQ_Natural Logarithm.
5.3.2 VQ_Log Base 10.
5.3.3 VQ_X to Power of Y.
5.3.4 VQ_Exp(e)^(X{n}).
5.3.5 VQ_Exp(10)^(X{n}).
5.3.6 VQ_Sine.
5.3.7 VQ_Cosine.

5.4 VQ Signal Processing.

5.4.1 VQ_Complex FFT.
5.4.2 VQ_Inverse Complex FFT.
5.4.3 VQ_Real FFT.
5.4.4 VQ_Inverse Real FFT.
5.4.5 VQ_Power Spectral Density.
5.4.6 VQ_Cross-Power Spectral Density.
5.4.7 VQ_Convolution.
5.4.7.1 VQ_Convolution with Offset.
5.4.8 VQ_Correlation.
5.4.9 VQ_Decimate.
5.4.10 VQ_Zero Fill.
5.4.11 VQ_Derivative X{n}
5.4.12 VQ_Integral X{n}

5.5 VQ Windows.

5.5.1 VQ_Window.
5.5.2 Window Coefficients Design.
5.5.3 VQ Windows Alternate VIs.
5.5.3.1 VQ_Hanning.
5.5.3.2 VQ_Hamming.
5.5.3.3 VQ_Triangle.
5.5.3.4 VQ_Blackman.
5.5.3.5 VQ_Exact Blackman.
5.5.3.6 VQ_Blackman Harris.
5.5.3.7 VQ_Flat Top.
5.5.3.8 VQ_Kaiser.

5.5.3.9 VQ_Cosine Tapered.
5.5.3.10 VQ_Force.
5.5.3.11 VQ_Exponential.

5.6 VQ Digital Filters.
5.6.1 VQ_IIR Filter.
5.6.1.1 IIR Direct Filter Coefficients Design.
5.6.2 VQ_IIR Cascade Filter.
5.6.2.1 IIR Cascade Filter Coefficients Design.
5.6.3 VQ_FIR Filter.
5.6.3.1 FIR Filter Coefficients Design.
5.6.4 VQ_LMS Adaptive Filter.

5.7 VQ Statistical.
5.7.1 VQ_RMS.
5.7.2 VQ_Mean.
5.7.3 VQ_Histogram.
5.7.4 VQ_Threshold Peak Detect.
5.7.5 VQ_Max & Min.
5.7.6 VQ_FSK Compare

5.8 VQ Vector Manipulation.
5.8.1 VQ_Subset.
5.8.2 VQ_Concatenate Vectors.
5.8.3 VQ_Mirror.
5.8.4 VQ_Reverse.
5.8.5 VQ_Real Bit Reversal.
5.8.6 VQ_Complex Bit Reversal.

5.9 VQ Matrices.
5.9.1 VQ_Matrix Addition.
5.9.2 VQ_Matrix Subtraction.
5.9.3 VQ_Matrix Scale.
5.9.4 VQ_Matrix Outer Product.
5.9.5 VQ_A .Dot. B.
5.9.6 VQ_Inverse Matrix.
5.9.7 VQ_Transpose.
5.9.8 VQ_Determinant.
5.9.9 VQ_AxX{n}.
5.9.10 VQ_AxB.
5.9.11 VQ_Solve Linear Equations.

5.10 VQ Conversion.
5.10.1 VQ_Build VQ Block From PQ Source.
5.10.2 VQ_Build VQ Block From Triggered PQ Source.
5.10.3 VQ_Build VQ Block From Analog Input.
5.10.4 VQ_Build Shifted VQ Block from PQ Source.

5.10.5 PQ Playback from VQ Source.
5.10.6 VQ_Rectangular to Polar.
5.10.7 VQ_Polar to Rectangular.

5.11 VQ Miscellaneous.
5.11.1 VQ_Get Vector from DSP.
5.11.2 VQ_ASYNC Get Vector from DSP.
5.11.3 VQ_Checked Vector Get.
5.11.4 VQ_Grab Vector.
5.11.5 VQ_Put Vector to DSP.

6.0 Queue Structure VIs.

6.1 PQ Case Structures.
6.1.1 PQ_Open Case Structure.
6.1.2 PQ_Terminate This Case.
6.1.3 PQ_Close Case Structure.

6.2 PQ Conditional Execute Structures.
6.2.1 PQ_Begin Conditional Execute.
6.2.2 PQ_End Conditional Execute.

6.3 PQ While Loop Structures.
6.3.1 PQ_GoTo While Loop Structure.
6.3.2 PQ_Begin While Loop Structure.
6.3.3 PQ_End While Loop Structure.

6.4 VQ Case Structures.
6.4.1 VQ_Open Case Structure.
6.4.2 VQ_Terminate This Case.
6.4.3 VQ_Close Case Structure.

6.5 VQ Conditional Execute Structures.
6.5.1 VQ_Begin Conditional Execute.
6.5.2 VQ_End Conditional Execute.

6.6 VQ While Loop Structures.
6.6.1 VQ_Begin While Loop Structure.
6.6.2 VQ_End While Loop Structure.

7.0 Supplemental SI Functions.
7.1 Construct Channel Array.
7.2 Boolean to Channel Array.
7.3 Application Launch.

**$$$

	LabVIEW 5.x
Sheldon_LV6->	All QuVIEW examples and demos for LabVIEW 6.x and LAbVIEW 7.x
SIC30DSP->	All low level QuX files, essential binaries for the host and DSP COFF files.
SI_Support->	
Docs->	All Sheldon Instruments documentation.
sidev->	precompiled binaries (drivers), command line examples, and all source code for custom develoment.
binaries->	Precompiled drivers for all SI hardware, along with precompiled examples for custom development.
drivers->	All SI hardware drivers for Windows, used for QuX and custom development.
fpgaloader->	FPGA loader utility.
sihw_apps->	Command line utitlities for board diagnosis used in custom development.
siddk->	Host side source code for custom development.
sidsp->	DSP side source code for custom development.

NOTES:
1) Notice the QuX path structure on the CD directly mirrors the path structure that must be followed whith the installed files on your system. For example, the '..\QuVIEW \root.mnu' folder and file under the '\LabVIEWx\menus\' folder on the CD, must be placed under the same '\..\LabVIEW\menus\' folder on your system.
2) The QuVIEW files for LabVIEW 7.x are identical as those supplied for LabVIEW 6.x, with the exception of the menu file, namely the 'root.mnu' file. Therefore, it is recommended that after all LabVIEW 6.x QuVIEW files are completely installed under LabVIEW 7.x, be sure to use the 'root.mnu' file specific to LabVIEW 7.x.

1.2 QuVIEW Files Installation.

***NOTE**:*
There are two steps that must be first performed in order for the SI hardware to operate correctly:
1) Drivers for all SI-DSP hardware. Please consult the '\DOCS\DRIVERS\README-DRIVERS.RTF' document for more details.
2) For certain SI hardware, the FPGA loader must be invoked in order to load the onboard logic. The FPGALoader must be run at least once before any DSP software can run; it is recommended that a batch file be executed just once everytime the computer is booted. Please consult the '\DOCS\SIDSP\FPGALoad\FPGALOAD.RTF' document for more details.

Once the drivers and onboard logic have been successfully loaded, QuVIEW installation may begin. Please copy the directories from the CD onto your hard drive as follows:

Source Directory	->	Destination Directory on Hard Disk
1. **\QuX\SIC30DSP**	->	**C:\SIC30DSP** (this folder must be on the C drive root)
2. **\QuX\QuVIEW\SHELDON_LVx**	->	**C:\..\SHELDON** (this folder can be placed anywhere)
3. **\QuX\QuVIEW\LABVIEWx\SI.LIB**	->	**C:\..\LABVIEWx\SI.LIB**
4. **\QuX\QuVIEW\LABVIEWx\MENUS \QuVIEW**	->	**C:\..\LABVIEWx\MENUS \QuVIEW**

***NOTE**:*
1) Files directly copied from CDs to hard drives retain their read-only attributes. This attribute needs to be removed once the files have been copied over to the hard drive. The quickest method is to highlight all files within a directory, right click with the mouse to invoke the 'Properties' menu item, and clear the adjacent checkbox box. This must be performed for all subdirectories copied onto the hard drive.
2) Once all of the QuVIEW libraries and demo files are installed, it is strongly advisable to add them to LabVIEW's search path defaults. Adding QuVIEW to the LabVIEW search path defaults enables their quick and automatic loading of applications as well as mass compiling, described in the next section.
3) Once QuVIEW files are added to LabVIEW's search paths, it is always advisable to perform a 'mass compile.' Mass compiling LabVIEW files ensures concurrency between the installed version of LabVIEW and the newly installed library files, described in the next section.
4) In order to view the QuVIEW icon palettes within the LabVIEW environment, select the 'QuVIEW' menu as 'default', described in the next section.
5) Before running any of the QuVIEW demos and apps, be sure that the Windows drivers are installed and the subsequent the FPGALoader has successfully loaded the onboard logic.

1.3 Adding QuVIEW to LabVIEW Paths.

After QuVIEW files have been installed, it is strongly advised to include them into LabVIEW's search path defaults. This can be done in the following ways:

1) Using the LabVIEW's graphical editor
LabVIEW 5.x.
Select 'Edit>>Preferences....>>Paths>>VI Search Path' which will invoke the 'Preferences' dialog box. Remove or unclick the 'Use Default' setting so that you are able to add both the '\..\LabVIEW\SI.LIB*' and 'C:\SIC30DSP*' paths to the overall list.

LabVIEW 6.x and LabVIEW 7.x.
Select 'Tools>>Options.....>>Paths>>VI Search Path' which will invoke the 'Preferences' dialog box. Remove or unclick the 'Use Default' setting so that you are able to add both the '\..\LabVIEW\SI.LIB*' and 'C:\SIC30DSP*' folder paths to the overall list.
NOTE:
The "/" characters must be added to the end of the selected paths in order to ensure that all items inside of this folder are accessed.*

2) Manually editing the LabVIEW.INI file.
Look for the 'viSearchPath' line, and append the complete paths for the '\..\LabVIEW \SI.LIB*' and 'C:\SIC30DSP*' folders. An example of this line is as follows:

viSearchPath="<topvi>:*;<foundvi>:\;<vilib>:*;<userlib>:*;<instrlib>:*;C:\Program Files\LABVIEW\si.lib*;C:\SIC30dsp*"

NOTE:
The "/" characters must be added to the end of the selected paths in order to ensure that all items inside of this folder are accessed, and each folder's path must be separated by a semicolon.*

Once the paths have been added, LabVIEW must be shut down and restarted in order for these changes to take effect. From now on, LabVIEW will automatically include these paths to load QuVIEW libraries for applications as well as mass compiling. A mass compile of all QuVIEW libraries and demos is also highly recommended in order to ensure concurrency between the installed version of LabVIEW and the newly installed QuVIEW files.

1.4 Viewing QuVIEW Menus Under LabVIEW.

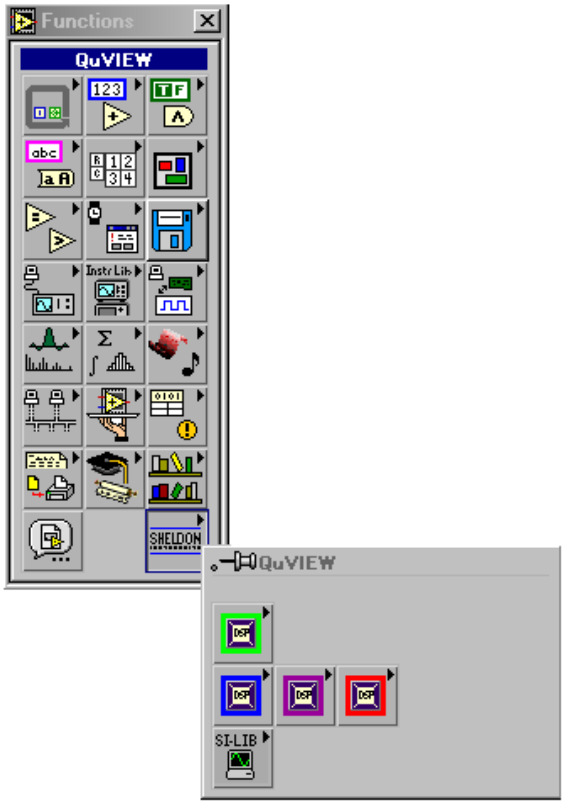
1) LabVIEW 5.x.
The newly installed QuVIEW library menus can be seen as regular icons within the 'Controls' and 'Function' palettes in LabVIEW 5.x by manually selecting the “QuVIEW” palettes under the ‘Edit>>Select Palette Set’ option.

2) LabVIEW 6.x and LabVIEW 7.x.
The newly installed QuVIEW library menus can be seen as regular icons by manually selecting the “QuVIEW” palettes after pressing the ‘Options’ button. The ‘Options’ button appears as the right most or the last of three buttons on the Functions Palette, and can only be selected graphically.

$$$

2.0 QuVIEW: DSP Real Time Acquisition, Signal Processing and Control.

The Sheldon Instruments' QuVIEW library VI menu is located at the bottom right corner of the Functions menu.



These VIs are used for implementing ultra high speed real time processing, with the added flexibility of using LabVIEW icons. A library of functions for real time acquisition, signal processing and control exists for algebra, calculus, digital filtering, Z transforms, frequency domain analysis, numerical analysis, waveform generation, and process control, which are constantly being expanded.

Two distinct methods of data processing within the DSP are implemented, and hence the libraries are divided in such a way so as to make this distinction more obvious. One method, categorized as "Point Queue" (VIs denoted with a "PQ_" prefix), is better suited for real time and continuous data processing in which data is processed on a point by point (sample by sample) basis and a result is returned before the next point is to be processed. These VIs are predominantly colored blue.

The other method, categorized as "Vector Queue" (VIs denoted with a "VQ_" prefix), is better suited for processing blocks of data points in which data is processed after the entire block has been received. These VIs are predominantly colored purple.

2.1 Building a QuVIEW Application.

The diagram for a QuVIEW application is very similar to that of a normal LabVIEW application. However, there are a few important issues which must be considered when constructing a VI that uses QuVIEW libraries.

Please refer to the file Tutorial.DOC in the same folder as this document. This document provides instructions for using the QuVIEW function libraries and building DSP programs.

2.2 Common Terminals to All QuVIEW VIs.

There are a number of terminals common to all QuVIEW VIs:

Board = Board Number of board installed. On all VIs, this is required terminal to wire.

Daughter Module = LabVIEW constant that defines the type of multifunction I/O module in place, it contains information about the daughter module converters' resolution and voltage scale.

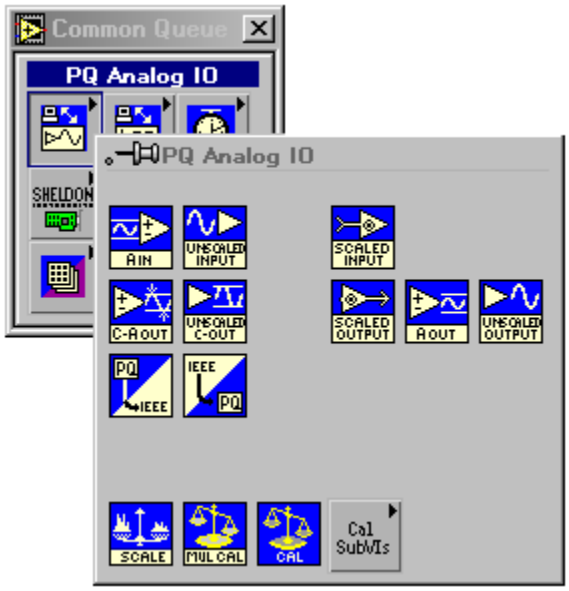
Error In = Error input cluster connection that is daisy chained with the "Error Out" cluster terminals of previous VIs in a data flow chain. Besides being used for error detection, wiring these terminals ensures deterministic data flow operation.

Error Out = Error output cluster connection that is daisy chained with the "Error In" cluster terminals of subsequent VIs in a data flow chain. Besides being used for error detection, wiring these terminals ensures deterministic data flow operation.

NOTE: *Input and Output connection terminal descriptions are color coded.* ***BLACK*** *is used to highlight input terminals, while* ***NAVY BLUE*** *highlights output terminals.*

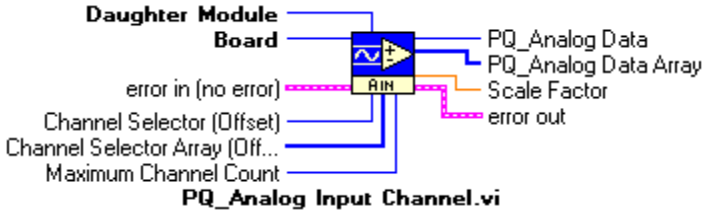
**$$$

3.1 Common Queue: Analog I/O.



3.1.1 PQ Analog Input Channel.

The PQ_Analog Input Channel.vi integrates the complete functionality associated with reading one or more analog inputs. This functionality includes reading the raw, bipolar signed binary value, and then scaling and converting it to a floating point value that represent its voltage based on the ADC's resolution and signal range.



Channel Selector (Offset) = LabVIEW constant used to select a single analog input of interest. Not to be used if an array is wired to the Channel Selector Array (Offset) terminal.

Channel Selector Array (Offset) = LabVIEW array of constants used to select an array of analog inputs. Convenient if many analog inputs are to be read simultaneously as it avoids the wiring of separate icons. If wired, it becomes the default channel selector as the Single Channel Selector (Offset) terminal is ignored.

Maximum Channel Count = LabVIEW constant used to limit the maximum number of elements in the 'Channel Selector Array' input. Allows the number of elements inside of the 'Channel Selector Array' to be altered in real time while a VI is running.

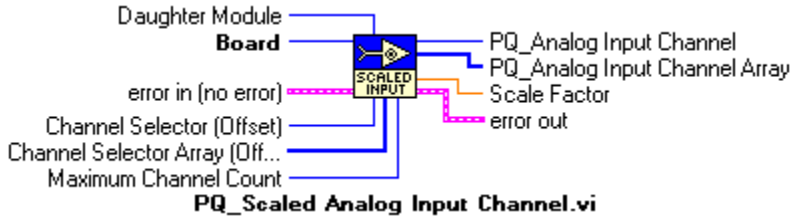
PQ_Analog Data = PQ output containing the actual scaled reading from a single selected analog input. If an array of channels is selected, only the first value of the array is returned.

PQ_Analog Data Array = PQ output array containing an array of scaled readings from the selected array of analog inputs. An empty array is returned if only the single input terminal is used.

Scale Factor = LabVIEW output used to read the exact analog voltage scaling factor.

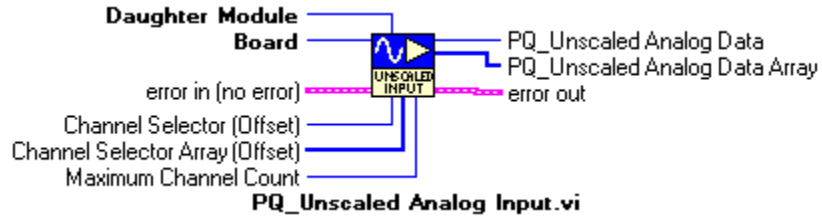
3.1.2 PQ_Scaled Analog Input Channel.

Same as PQ_Analog Input Channel.vi, except it uses several subVIs to perform same function, and therefore is not efficient. It is not recommended for use.



3.1.3 PQ_Unscaled Analog Input.

The PQ_Unscaled Analog Input.vi simply reads one or more analog inputs as raw, bipolar, signed binary values, no scaling or calibration takes place.



Channel Selector (Offset) = LabVIEW constant used to select a single analog input of interest. Not to be used if an array is wired to the Channel Selector Array (Offset) terminal.

Channel Selector Array (Offset) = LabVIEW array of constants used to select an array of analog inputs. Convenient if many analog inputs are to be read simultaneously as it avoids the wiring of separate icons. If wired, it becomes the default channel selector as the Single Channel Selector (Offset) terminal is ignored.

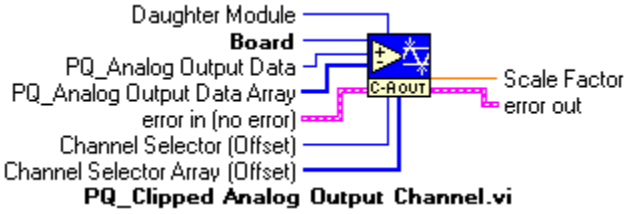
Maximum Channel Count = LabVIEW constant used to limit the maximum number of elements in the 'Channel Selector Array' input. Allows the number of elements inside of the 'Channel Selector Array' to be altered in real time while a VI is running.

PQ_Unscaled Analog Data = PQ output containing the actual unscaled binary reading from a single selected analog input. If an array of channels is selected, only the first value of the array is returned.

PQ_Unscaled Analog Data Array = PQ output array containing an array of unscaled binary readings from the selected array of analog inputs. An empty array is returned if only the single input terminal is used.

3.1.4 PQ_Clipped Analog Output Channel.

The PQ_Clipped Analog Output Channel.vi integrates the complete functionality associated with writing one or more analog outputs. This functionality includes writing the voltage represented as a scaled floating point value, and then unscaling and converting it to the raw, bipolar signed binary value based on the DAC's resolution and signal range, as well as clipping the limits so as to avoid wrapping.



Channel Selector (Offset) = LabVIEW constant used to select a single analog output of interest. Not to be used if an array is wired to the Channel Selector Array (Offset) terminal.

Channel Selector Array (Offset) = LabVIEW array of constants used to select an array of analog outputs. Convenient if many analog outputs are to be written simultaneously as it avoids the wiring of separate icons. If wired, it becomes the default channel selector terminal as the Single Channel Selector (Offset) terminal is ignored.

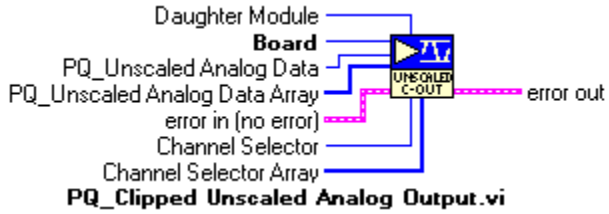
PQ_Analog Output Data = PQ input containing the actual scaled value to be written to a single selected analog output. If an array of channels is selected, this terminal is ignored.

PQ_Analog Output Data Array = PQ input array containing an array of scaled values to be written to an array of analog outputs. Should be used if the Channel Selector Array terminal is used, since both array sizes must be the same and have a one to one correspondence.

Scale Factor = LabVIEW output used to read the exact analog voltage scaling factor.

3.1.5 PQ_Clipped Unscaled Analog Output.

The PQ_Clipped Unscaled Analog Output.vi simply writes one or more analog outputs as raw, bipolar, signed binary values, no scaling or calibration takes place except for the clipping of the limits so as to avoid wrapping.



Channel Selector (Offset) = LabVIEW constant used to select a single analog output of interest. Not to be used if an array is wired to the Channel Selector Array (Offset) terminal.

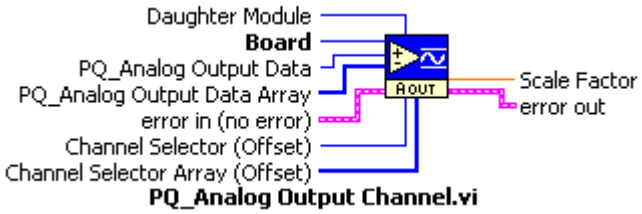
Channel Selector Array (Offset) = LabVIEW array of constants used to select an array of analog outputs. Convenient if many analog outputs are to be written simultaneously as it avoids the wiring of separate icons. If wired, it becomes the default channel selector as the Single Channel Selector (Offset) terminal is ignored.

PQ_Unscaled Analog Data = PQ input containing the actual unscaled signed binary value to be written to a single selected analog output. If an array of channels is selected, this terminal is ignored.

PQ_Unscaled Analog Data Array = PQ input array containing an array of unscaled signed binary values to be written to an array of analog outputs. Should be used if the Channel Selector Array terminal is used, since both array sizes must be the same and have a one to one correspondence.

3.1.6 PQ_Analog Output Channel.

The PQ_Analog Output Channel.vi integrates the complete functionality associated with writing one or more analog outputs. This functionality includes writing the voltage represented as a scaled floating point value, and then unscaling and converting it to the raw, bipolar signed binary value based on the DAC's resolution and signal range.



Channel Selector (Offset) = LabVIEW constant used to select a single analog output of interest. Not to be used if an array is wired to the Channel Selector Array (Offset) terminal.

Channel Selector Array (Offset) = LabVIEW array of constants used to select an array of analog outputs. Convenient if many analog outputs are to be written simultaneously as it avoids the wiring of separate icons. If wired, it becomes the default channel selector terminal as the Single Channel Selector (Offset) terminal is ignored.

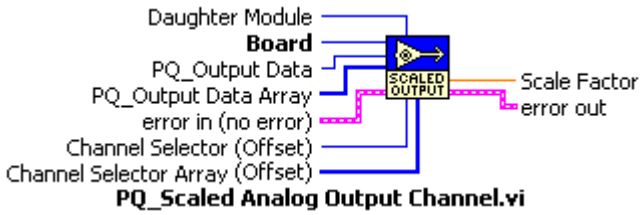
PQ_Analog Output Data = PQ input containing the actual scaled value to be written to a single selected analog output. If an array of channels is selected, this terminal is ignored.

PQ_Analog Output Data Array = PQ input array containing an array of scaled values to be written to an array of analog outputs. Should be used if the Channel Selector Array terminal is used, since both array sizes must be the same and have a one to one correspondence.

Scale Factor = LabVIEW output used to read the exact analog voltage scaling factor.

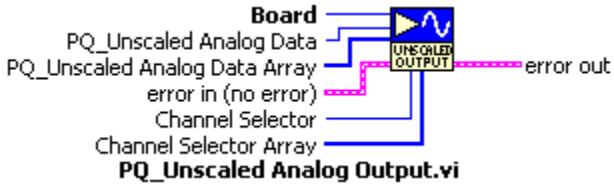
3.1.7 PQ_Scaled Analog Output Channel.

Same as PQ_Analog Output Channel.vi, except it uses several subVIs to perform the same function, and therefore is not efficient. It is not recommended for use.



3.1.8 PQ_Unscaled Analog Output.

The PQ_Unscaled Analog Output.vi simply writes one or more analog outputs as raw, bipolar, signed binary values, no scaling or calibration takes place.



Channel Selector (Offset) = LabVIEW constant used to select a single analog output of interest. Not to be used if an array is wired to the Channel Selector Array (Offset) terminal.

Channel Selector Array (Offset) = LabVIEW array of constants used to select an array of analog outputs. Convenient if many analog outputs are to be written simultaneously as it avoids the wiring of separate icons. If wired, it becomes the default channel selector as the Single Channel Selector (Offset) terminal is ignored.

PQ_Unscaled Analog Data = PQ input containing the actual unscaled signed binary value to be written to a single selected analog output. If an array of channels is selected, this terminal is ignored.

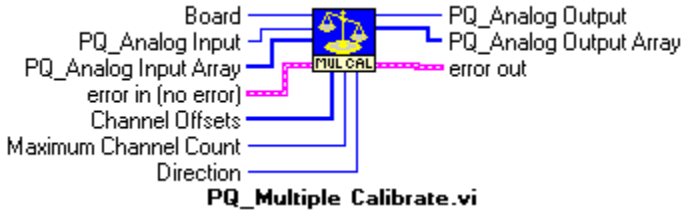
PQ_Unscaled Analog Data Array = PQ input array containing an array of unscaled signed binary values to be written to an array of analog outputs. Should be used if the Channel Selector Array terminal is used, since both array sizes must be the same and have a one to one correspondence.

3.1.9 PQ_Calibrate.

The PQ_Calibrate.vi performs gain and offset calibration to analog I/Os. The calibration error values are contained in a file with the *.TBL extension. For advanced users and is not recommended for use.

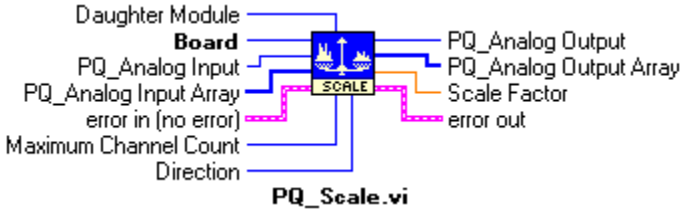
3.1.10 PQ_Multiple Calibrate.

The PQ_Multiple Calibrate.vi performs gain and offset calibration on an array of analog I/Os. The calibration error values are contained in a file with the *.TBL extension. For advanced users and is not recommended for use.



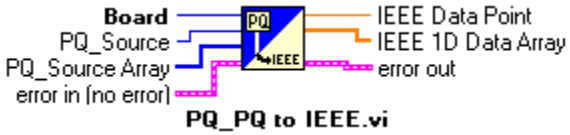
3.1.11 PQ_Scale.

The PQ_Scale.vi performs a conversion between binary and voltage values associated with ADCs and DACs. For advanced users and is not recommended for use.



3.1.12 PQ_PQ Source to IEEE.

The PQ_PQ Source to IEE.vi serves to transfer any PQ value or array of PQ values from the DSP domain to the LabVIEW/host domain. The floating point PQ values are read from the DSP and converted to the IEEE floating point format supported by LabVIEW. This function is mostly performed with minimal DSP intervention.



PQ_Input = PQ input source to be transferred to the LabVIEW domain for immediate use. Not to be used if an array is wired

PQ_Input Array = PQ input source array to be transferred to the LabVIEW

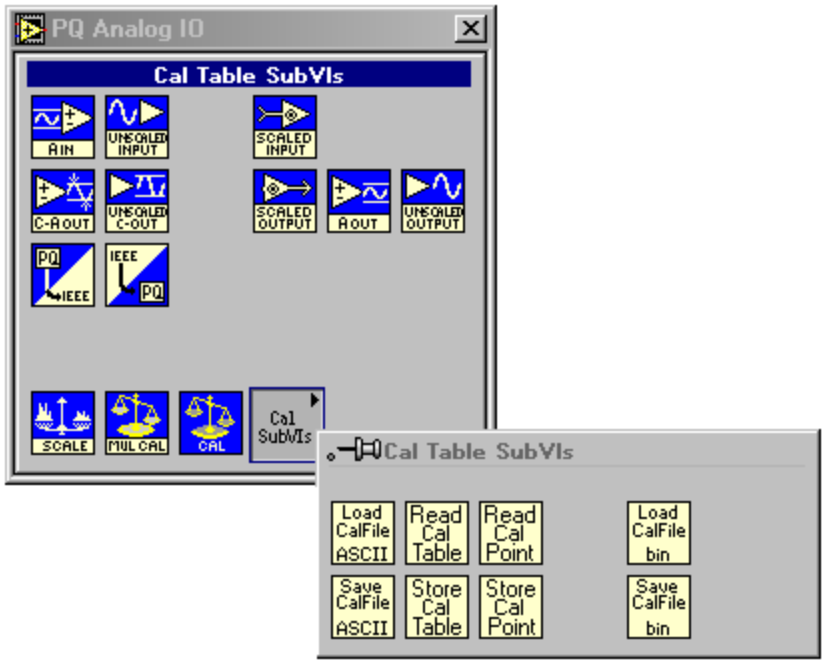
domain for immediate use. Convenient if an array of variables is used as it avoids the wiring of separate icons. If wired, it becomes the default terminal used, and the single variable terminal is ignored.

IEEE Data Point = LabVIEW output variable represented as a single precision floating point number. Not to be used if an array is wired.

IEEE 1D Data Array = LabVIEW output array variable representing an array of single precision floating point numbers. If an input array was used, then the output array will be correspondingly used.

3.1.13 PQ Analog I/O: Calibration SubVIs.

A set of subVIs used for calibrating gain and offset errors present on analog I/O channels. The gain and offset errors are contained inside a file with a *.TBL extension which resides inside of the \SIC30DSP directory. For advanced users and not intended for general use.



3.1.13.1 Load Cal File.

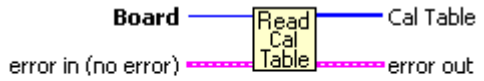
Loads a Calibration Table file. For advanced users and not intended for general use.

Path to Cal Table = Path to Calibration Table file.

Cal Table = An array of values read from the Calibration file.

3.1.13.2 Read Cal Table.

Reads from a Calibration Table already residing in LabVIEW/host memory. For advanced users and not intended for general use.

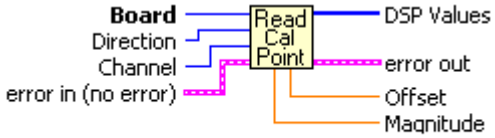


Read Current Calibration Table.vi

Cal Table = An array of calibration values.

3.1.13.3 Read Cal Point.

Reads a single calibration point for a given analog channel. For advanced users and not intended for general use.



Read Single Cal Value.vi

Direction = Denotes if the calibration is to apply to an analog input or an analog output.

Channel = Channel number of the analog I/O channel of interest.

Offset = Offset error value to be applied to a particular analog channel of interest.

Magnitude = Magnitude error value to be applied to a particular analog channel of interest.

DSP Values = An array containing the Offset and Magnitude error values of a particular analog I/O channel of interest, converted to DSP format.

3.1.13.4 Store Cal Table.

Stores a Calibration table into LabVIEW/host memory. For advanced users and not intended for general use.

Store Current Calibration Table.vi

Cal Table = Array of Cal Table values

3.1.13.5 Save Cal File.

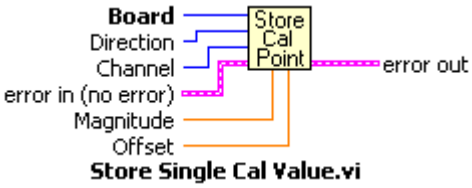
Saves a Calibration table into a file. For advanced users and not intended for general use.

Path to Cal Table = Path to Calibration Table file.

Cal Table = An array of values to be saved into the Calibration file.

3.1.13.6 Store Cal Point.

Stores a calibration value of a single analog channel. For advanced users and not intended for general use.



Direction = Denotes if the calibration is to apply to an analog input or an analog output.

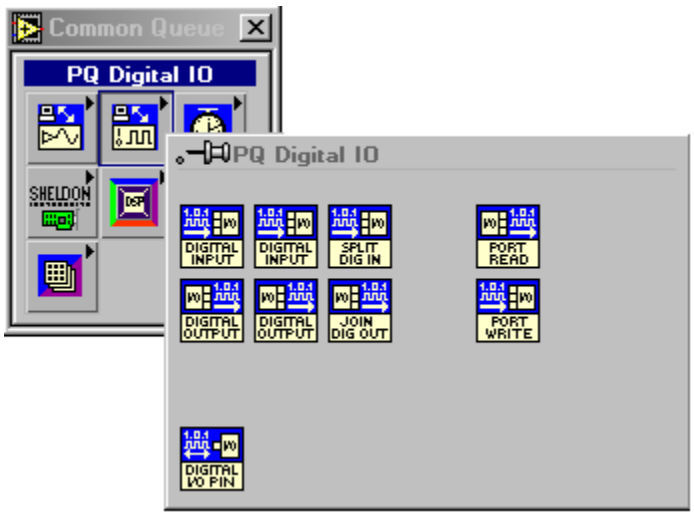
Channel = Channel number of the analog I/O channel of interest.

Offset = Offset error value to be applied to a particular analog channel of interest.

Magnitude = Magnitude error value to be applied to a particular analog channel of interest.

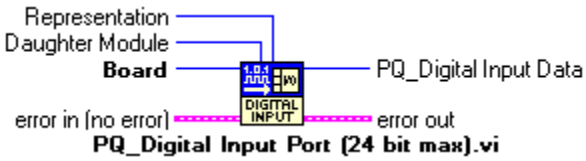
DSP Values = An array containing the Offset and Magnitude error values of a particular analog I/O channel of interest, converted to DSP format.

3.2 Common Queue: Digital I/O.



3.2.1 PQ_Digital Input Port (24 bit max).

The PQ_Digital Input Port.vi reads up to 24 bits of raw binary data present on the digital I/O port when it is configured as an input.

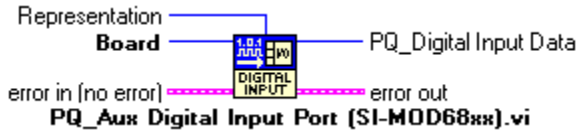


Representation = LabVIEW input, selects to read either signed or unsigned binary data from the digital I/O port; defaulted to read unsigned integer values.

PQ_Digital Input Data = PQ output, up to 24 bits of binary data read from the digital I/O port, represented as a floating point value.

3.2.2 PQ_Aux Digital Input Port (SI-MOD68xx).

The PQ_Aux Digital Input Port (SI-MOD68xx).vi reads up to 4 bits of raw binary data present on the auxiliary digital I/O port of the SI-MOD68xx module when it is configured as an input.

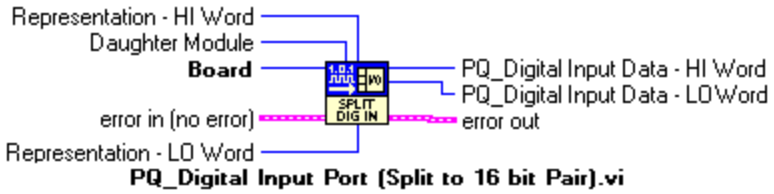


Representation = LabVIEW input, selects to read either signed or unsigned binary data from the digital I/O port; defaulted to read unsigned integer values.

PQ_Digital Input Data = PQ output, up to 4 bits of binary data read from the auxiliary digital I/O port, represented as a floating point value.

3.2.3 PQ Digital Input Port (Split to 16 bit Pair).

The PQ_Digital Input Port (Split into 16 bit Pair).vi reads a single 32 bit port as raw binary data, and separates the reading into a pair of 16 bit ports when it is configured as an input. This split is necessary since the DSP is only able to represent a 24 bit integer when that value is encoded as a PQ compatible floating point value.


Representation - HI Word
Daughter Module
Board
error in (no error)
Representation - LO Word
SPLIT DIG IN
PQ_Digital Input Data - HI Word
PQ_Digital Input Data - LO Word
error out
PQ_Digital Input Port (Split to 16 bit Pair).vi

Representation - HI Word = LabVIEW input, selects to read either signed or unsigned binary data from the upper half 16 bit word of a 32 bit digital I/O port; defaulted to read unsigned integer values.

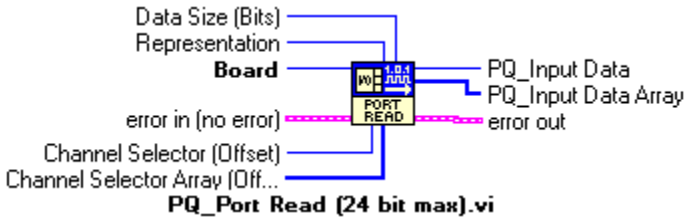
Representation - LO Word = LabVIEW input, selects to read either signed or unsigned binary data from the lower half 16 bit word of a 32 bit digital I/O port; defaulted to read unsigned integer values.

PQ_Digital Input Data - HI Word = PQ output, up to 16 bits of binary data read from the upper half 16 bit word of a 32 bit digital I/O port, represented as a floating point 'PQ' value.

PQ_Digital Input Data - LO Word = PQ output, up to 16 bits of binary data read from the lower half 16 bit word of a 32 bit digital I/O port, represented as a floating point 'PQ' value.

3.2.4 PQ Port Read (24 bit max).

The PQ_Port Read.vi reads up to 24 bits of raw binary data any within the DSP memory space. Used as a building block for other functions.



Data Size (Bits) = LabVIEW input, specifies the width of the port in bits ranging from 1 to 24 bits.

Representation = LabVIEW input, selects to read either signed or unsigned binary data from the digital I/O port; defaulted to read unsigned integer values.

Channel Selector (Offset) = LabVIEW input, specifies the actual location of the port to be read in the DSP's memory space.

Channel Selector Array (Offset) = LabVIEW input array, which specifies an

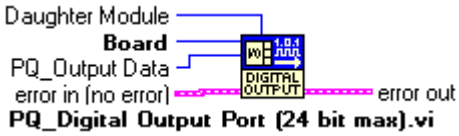
array of actual locations of ports to be read in the DSP's memory space.

PQ_Input Data = PQ output, up to 24 bits of binary data read from the digital port, represented as a floating point value.

PQ_Input Data Array = PQ output array, up to 24 bits of a binary data read from an array of digital port, represented as floating point values.

3.2.5 PQ_Digital Output Port (24 bit max).

The PQ_Digital Output Port.vi writes up to 24 bits of raw binary data to the digital I/O port when it is configured as an output.



PQ_Output Data = PQ input, up to 24 bits of binary data can be written to the digital I/O port, represented as a floating point value.

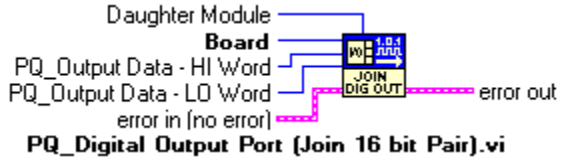
3.2.6 PQ_Aux Digital Output Port (SI-MOD68xx).

The PQ_Aux Digital Output Port (SI-MOD68xx).vi writes up to 4 bits of raw binary data to the auxiliary digital I/O port when it is configured as an output.

PQ_Output Data = PQ input, up to 4 bits of binary data can be written to the auxiliary digital I/O port, represented as a floating point value.

3.2.7 PQ_Digital Output Port (Join 16 bit Pair).

The PQ_Digital Output Port (Join 16 bit Pair).vi joins a pair of 16 bit values, and joins them to form a single 32 bit value to write to the digital I/O port when it is configured as an output. This concatenation is necessary since the DSP is only able to represent a 24 bit integer when that value is encoded as a PQ compatible floating point value.

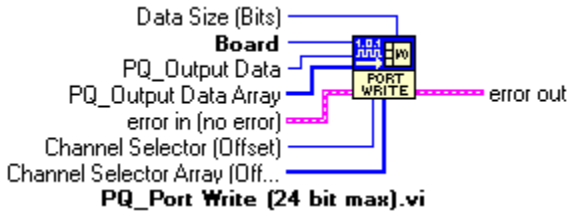


PQ_Output Data - HI Word = PQ input, up to 16 bits of binary data can be written to the upper half 16 bit word of a 32 bit digital I/O port, represented as a floating point 'PQ' value.

PQ_Output Data - LO Word = PQ input, up to 16 bits of binary data can be written to the lower half 16 bit word of a 32 bit digital I/O port, represented as a floating point 'PQ' value.

3.2.8 PQ_Port Write (24 bit max).

The PQ_Port Write.vi writes up to 24 bits of raw binary data to any within the DSP memory space. Used as a building block for other functions.



Data Size (Bits) = LabVIEW input, specifies the width of the port in bits ranging from from 1 to 24 bits.

PQ_Output Data = PQ input, up to 24 bits of binary data can be written to the digital port, represented as a floating point value. Not to be used if an array terminal is wired.

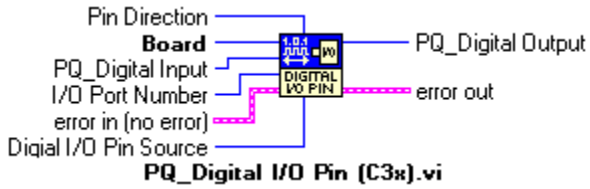
PQ_Output Data Array = PQ input array, up to 24 bits of a binary data can be written to an array of digital ports, represented as floating point values. Convenient if an array of variables is used as it avoids the wiring of separate icons. If wired, it becomes the default terminal used, and the single variable terminal is ignored.

Channel Selector (Offset) = LabVIEW input, specifies the actual location of the port to write in the DSP's memory space. Not to be used if an array terminal is wired.

Channel Selector Array (Offset) = LabVIEW input array, specifies an array of actual port locations to write in the DSP's memory space. If the array terminal for sourcing the data was used, then the Channel Selector Array should be correspondingly used.

3.2.9 PQ_Digital I/O Pin (C3x).

The PQ_Digital I/O Pin (C3x).vi makes use of the DSP's peripheral port as a general purpose digital I/O port, where each pin's direction may be individually controlled.



Pin Direction = LabVIEW input, specifies if a particular line will operate as an input or an output.

PQ_Digital Input = PQ Input, unscaled value to be writen if the pin of interest is configured as an output.

I/O Port Number = LabVIEW input, specifies one of two ports where the digital I/O pin is located. Only valid for those DSP's that have more than a single peripheral port; otherwise defaulted to the only port available.

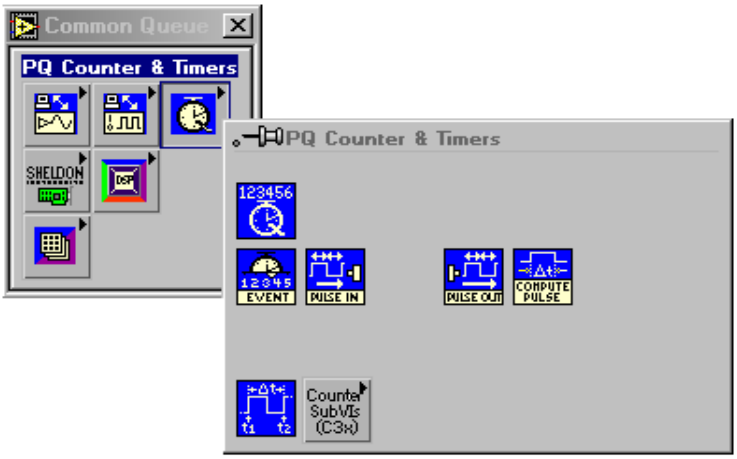
Digital I/O Pin Source = LabVIEW input, selects a specific pin of the DSP's port for use as a digital I/O.

PQ_Digital Output = PQ output representing the value present on a requested digital I/O pin.

NOTE: Only valid wth C33 DSPs.

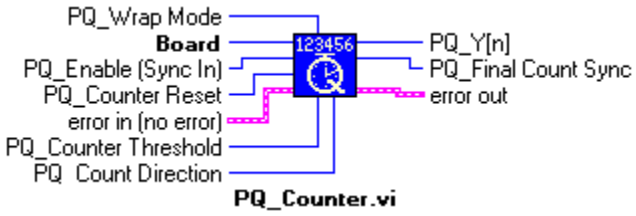
3.3 Common Queue: PQ Counters & Timers.

These functions are internal to the DSP, and make use of its peripheral port for unique digital sampling functions.



3.3.1 PQ_Counter.

The PQ_Counter.vi function operates in much the same way as an event counter, where the events can be defined by boolean pulses derived from any one of the comparison VIs.



PQ_Wrap Mode = PQ input, used to control if the counter must autowrap to the reset state when the count reaches the threshold value.

<=0 = When set to a 'PQ Zero' value or less, the counter is reset or wrapped when the threshold is reached as long as there is a tick source (default).

nonzero = Otherwise, when set to a 'PQ Nonzero' value the counter output PQ_Y[n] is latched at the threshold value, until the 'PQ_Counter Reset' terminal is cleared with a 'PQ Zero' value fed into it.

PQ_Enable (Sync In) = PQ input that determines of the counts are enabled or not

<=0 = when set to a 'PQ Zero' value or less, the counter is enabled to increment or decrement (default).

nonzero = when set to a 'PQ Nonzero' value, the counter is detained with the current value and NOT cleared.

PQ_Counter Reset = PQ input, used to clear the counter

<=0 = when set to a 'PQ Zero' value or less, the counter increments or decrements normally (default).

nonzero = when set to a 'PQ Nonzero' value, the counter is cleared irrespective of the state of the 'PQ_Enable' line.

PQ_Counter Threshold = PQ input constant that determines the last count value before wrapping or latching.

<=0 = the counter increments or decrements freely until 2^32 approaches, after which it wraps to a 'PQ Zero' value or latches, as set by the 'PQ_Wrap Mode' input terminal, (default).

nonzero = the counter increments or decrements to this PQ value before wrapping or latchingmust be always positive.

PQ Count Direction = PQ input that determines the direction of the count.

<=0 = When set to a 'PQ Zero' value the counter increments to +threshold, (default).

nonzero = When set to a 'PQ Nonzero' value, the counter decrementsto -threshold.

PQ_Y[n] = PQ output that returns actual count value.

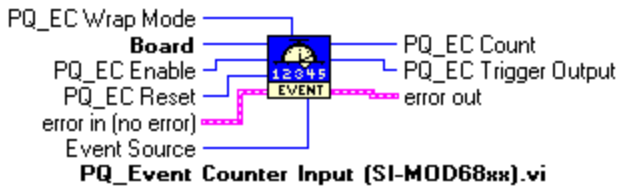
PQ_Final Count Sync = PQ output used to indicate the state of the counter output, useful for synchronizing events.

0 = The counter output has reached its final value, and returns a 'PQ zero' value

1 = The counter is busy and returns a 'PQ One' while counting or reset.

3.3.2 PQ_Event Counter Input (SI-MOD68xx).

The PQ_Event Counter Input (SI-MOD68xx).vi function is very flexible and especially useful for managing a variety of clocking and event schemes to trigger a variety of operations.



PQ_EC Wrap Mode = PQ input, used to control if the EC counter must autowrap to the reset state when the count reaches the threshold value.

<=0 = When set to a 'PQ Zero' value or less, the counter is reset or wrapped when the threshold is reached as long as there is a tick source (default).

nonzero = Otherwise, when set to a 'PQ Nonzero' value the counter output PQ_Y[n] is latched at the threshold value, until the 'PQ_Counter Reset' terminal is cleared with a 'PQ Zero' value fed into it.

PQ_EC Enable = PQ input that determines of the counts are enabled or not.

<=0 = when set to a 'PQ Zero' value or less, the counter is enabled to increment or decrement (default).

nonzero = when set to a 'PQ Nonzero' value, the counter is detained with the current value and NOT cleared.

PQ_EC Reset = PQ input, used to clear the EC counter.

<=0 = when set to a 'PQ Zero' value or less, the counter increments or decrements normally (default).

nonzero = when set to a 'PQ Nonzero' value, the counter is cleared irrespective of the state of the 'PQ_Enable' line.

Event Source = LabVIEW input, selects one of four Event Counter circuits.

PQ_EC Count = PQ output that returns actual count value.

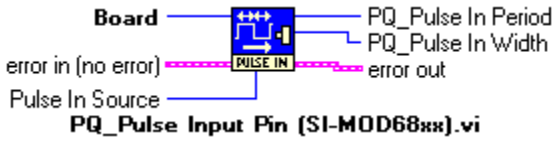
PQ_EC Trigger Output = PQ output used to indicate the state of the counter output, useful for synchronizing events. This signal may also be used as a software source to other devices (reverse from bit reading).

0 = The counter output has reached its final value, and returns a 'PQ zero' value.

1 = The counter is busy and returns a 'PQ One' while counting or reset.

3.3.3 PQ_Pulse Input Pin (SI-MOD68xx).

The PQ_Pulse Input Pin (SI-MOD68xx).vi is an extremely powerful function that operates as a frequency counter on one of the lines available on the auxiliary digital I/O ports, present on the SI-MOD68xx module.



Pulse In Source = LabVIEW input, selects one of the available pulse input pins present on the auxiliary digital I/O port.

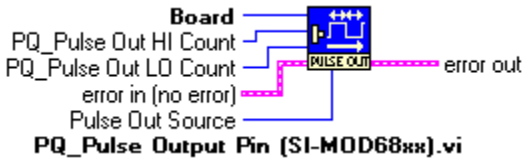
PQ_Pulse In Period = PQ output that returns the last overall period that occured between consecutive positive edges derived from the external TTL level pulse train input. The period is relative to the number of H clock periods sourced from the DSP carrier card.

PQ_Pulse In Width = PQ output that returns the number of H clock periods that occured between every rising and falling edge of an external TTL level pulse train. The period is relative to the number of H clock periods sourced from the DSP carrier card.

NOTE: *The H Clock rate varies depending upon the particular DSP installed on the carrier card. Currently, all Sheldon Instruments hardware supports the following H clock rates:*
1) SI-C31DSP-PCI cards. The H clock rate for the C31 DSP ranges from 20Mhz to 30Mhz.
2) SI-C33DSP-PCI cards. The H clock rate for the C33 DSP is fixed at 37.5Mhz.
3) SI-C67xDSP-PCI cards. The H clock rate for the C67x DSP is fixed at 37.5Mhz.

3.3.4 PQ_Pulse Output.

The 'PQ Pulse Output.vi' is used to generate TTL level output pulse trains, used in PWM applications. May be optionaly used in conjunction with the 'PQ_Pulse Width Computation.vi'. Please consult the TI DSP's user manual for mor details concerning the H clock characteristics.



PQ_Pulse Out HI Count = PQ input, Width/Duration of the output pulse asserted HI (the number of periods between the rising and falling edges), relative to the number of DSP H clock periods. The period is relative to the number of H clock periods sourced from the DSP carrier card.

PQ_Pulse Out LO Count = PQ input, Width/Duration of the output pulse asserted LO (the number of periods between the falling and rising falling edges), relative to the number of DSP H clock periods. The period is relative to the number of H clock periods sourced from the DSP carrier card.

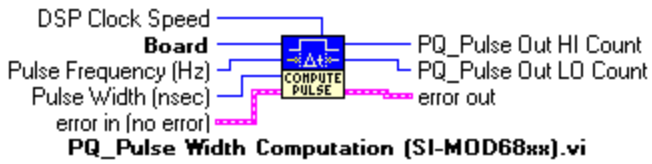
NOTE: *The sm of these two values is equivalent to the overall pulse period.*

Pulse Out Source = LabVIEW input, selects one of the available pulse output pins present on the auxiliary digital I/O port.

NOTE: *The H Clock rate varies depending upon the particular DSP installed on the carrier card. Currently, all Sheldon Instruments hardware supports the following H clock rates:*
1) SI-C31DSP-PCI cards. The H clock rate for the C31 DSP ranges from 20Mhz to 30Mhz.
2) SI-C33DSP-PCI cards. The H clock rate for the C33 DSP is fixed at 37.5Mhz.
3) SI-C67xDSP-PCI cards. The H clock rate for the C67x DSP is fixed at 37.5Mhz.

3.3.5 PQ_Pulse Width Computation.

Computes the conversion values to be downloaded to PQ_Pulse Output.vi.



DSP Clock Speed (Mhz) = LabVIEW input to select the time period, based on the DSP Clock speed.

Pulse Width (nsec) = LabVIEW input to define output pulse HI asserted level duration in nanoseconds.

Pulse Frequency (Hz) = LabVIEW input to define the overall pulse frequency in hertz.

PQ_Pulse Out HI Count = PQ output, Width/Duration of the output pulse asserted HI (the number of periods between the rising and falling edges), relative to the number of DSP H clock periods. The period is relative to the number of H clock periods sourced from the DSP carrier card.

PQ_Pulse Out LO Count = PQ output, Width/Duration of the output pulse asserted LO (the number of periods between the falling and rising falling edges), relative to the number of DSP H clock periods. The period is relative to the number of H clock periods sourced from the DSP carrier card.

NOTE: *The sm of these two values is equivalent to the overall pulse period.*

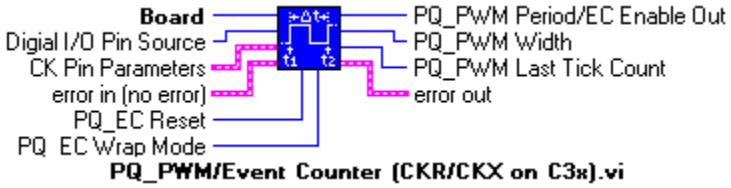
NOTE: *The H Clock rate varies depending upon the particular DSP installed on the carrier card. Currently, all Sheldon Instruments hardware supports the following H clock rates:*
1) SI-C31DSP-PCI cards. The H clock rate for the C31 DSP ranges from 20Mhz to 30Mhz.
2) SI-C33DSP-PCI cards. The H clock rate for the C33 DSP is fixed at 37.5Mhz.
3) SI-C67xDSP-PCI cards. The H clock rate for the C67x DSP is fixed at 37.5Mhz.

3.3.6 PQ_PWM/Event Counter (CLKx on C3x).

The PQ_PWM/Event Counter (CLKx on C3x) is an extremely powerful function that operates as a frequency counter on one of the four serial port clock pins of the C3x DSP.

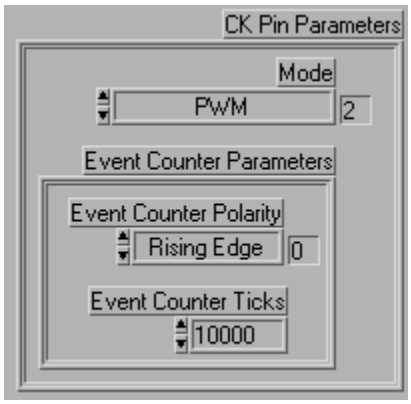
It returns a PQ_Last Pulse Width value and a PQ_Last Edge Time value based on the C3x's internal free running Timer 0 clock. In essence, the PQ_Last Pulse Width is the number of Timer 0 counts that occur between every rising or falling edge of an external TTL level pulse train. In order to compute the actual frequency of the external pulse train, divide the PQ_Last Pulse Width value into one fourth the C3x's clock speed. For example, if the C3x's clock speed is rated at 60Mhz, divide 15Mhz by the PQ_Last Pulse Width value.

The PQ_Last Edge Time simply returns the last value of the C3x's internal Timer 0 value, as opposed to a net difference provided by the PQ_Last Pulse Width output. Please consult the C3x User's Guide for further details about the C3x's internal architecture.



Digital I/O Pin Source (0-3) = LabVIEW input, ranging from 0 to 1 for a C31 and C32 or 0 to 3 for a C30, that determines which of the four serial clock port pins of the C3x is to be the PWM input.

CK Pin Paramerters = LabVIEW cluster input, defines the operation of the C3x's clock pins.



Mode = LabVIEW input, selects the operation mode of the C3x's clock pin I/Os.

0 = No operation.
1 = Digital I/O.
2 = PWM or Pulse input.
3 = Event Counter.

Event Counter Parameter = LabVIEW subcluster input, used to define the operation of the Event Counter.

Event Counter Polarity = LabVIEW input, selects the polarity of the Event Counter tick source.

0 = selects the tick source to increment the Event Counter on the positive/rising edges.

1 = selects the tick source to increment the Event Counter on the negative/falling edges.

Event Counter Ticks = LabVIEW input, defines the threshold or count limit for the Event counter.

PQ_EC Reset = PQ input, used to clear the PWM/Event Counter.

<=0 = when set to a 'PQ Zero' value or less, the counter increments or decrements normally (default).

nonzero = when set to a 'PQ Nonzero' value, the counter is cleared irrespective of the state of the 'PQ_Enable' line.

PQ_EC Wrap Mode = PQ input, used to control if the EC counter must

autowrap to the reset state when the count reaches the threshold value.

<=0 = When set to a 'PQ Zero' value or less, the counter is reset or wrapped when the threshold is reached as long as there is a tick source (default).

nonzero = Otherwise, when set to a 'PQ Nonzero' value the counter output PQ_Y[n] is latched at the threshold value, until the 'PQ_Counter Reset' terminal is cleared with a 'PQ Zero' value fed into it.

PQ_PWM Period/EC Enable Out = PQ output whose value depends on the operation mode selected for this C3x digital I/O pin.

PQ_PWM Period = If selected to operate as a pulse input device, the PQ ouputs returns the last overall period relative to the number of Timer 0 counts that occured between the edges of an external TTL level pulse train

Event Counter Enable Out = If selected to operate as an Event Counter, the PQ output used to indicate the state of the counter output, useful for synchronizing events. This signal may also be used as a software source to other devices (reverse from bit reading).

0 = The counter output has reached its final value, and returns a 'PQ zero' value.

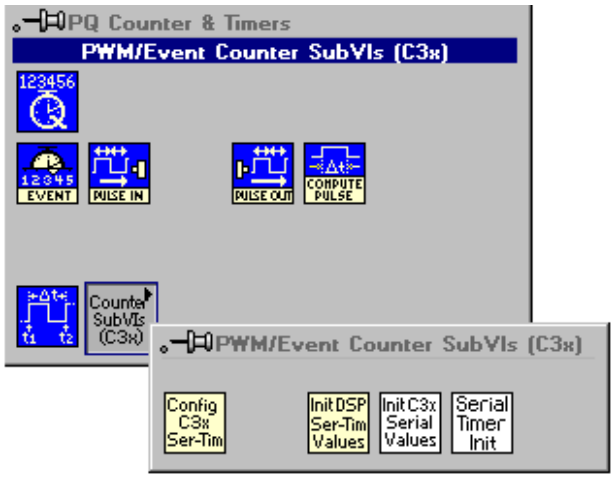
nonzero = The counter is busy and returns a 'PQ One' while counting or reset.

PQ_PWM Width = If the C3x digital I/O pin is selected to operate as a pulse input device, this PQ output returns the number of Timer 0 counts that occur between every rising or falling edge of an external TTL level pulse train. Not used if the pin is selected to operate as digital I/O or an Event Counter.

PQ_PWM Last Tick Count = If the C3x digital I/O pin is selected to operate as a pulse input device, PQ output that returns the last value of the C3x's internal Timer 0 value (Hx clock/2 value). Not used if the pin is selected to operate as digital I/O or an Event Counter.

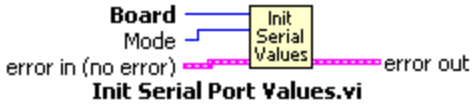
NOTE: *The H Clock rate varies depending upon the particular DSP installed on the carrier card. Currently, all Sheldon Instruments hardware supports the following H clock rates:*
1) SI-C31DSP-PCI cards. The H clock rate for the C31 DSP ranges from 20Mhz to 30Mhz.
2) SI-C33DSP-PCI cards. The H clock rate for the C33 DSP is fixed at 37.5Mhz.
3) SI-C67xDSP-PCI cards. The H clock rate for the C67x DSP is fixed at 37.5Mhz.

3.3.7. PWM/Event Counter SubVIs (C3x).



3.3.7.1 Init Serial Port Values.

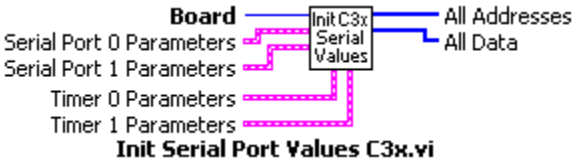
Initializes the C3x's serial port, whose pins are defaulted to be used as general purpose digital IO port or it can be reconfigured to be used to implement digital counters.



Mode = LabVIEW input, determines of the serial port is to be used for general purpose digital IO or if it is to be used as a frequency counter. To be used in conjunction with the PQ_Digital IO VIs or the PQ_Counter VIs.

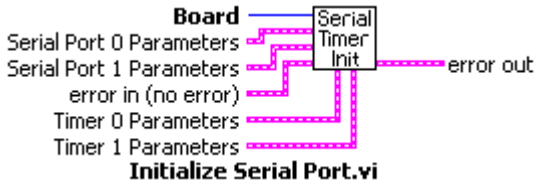
3.3.7.2 Init Serial Port Values C3x.

Initialized the C3x's serial port, whose pins are defaulted to be used as general purpose digital IO port or it can be reconfigured to be used to implement digital counters.



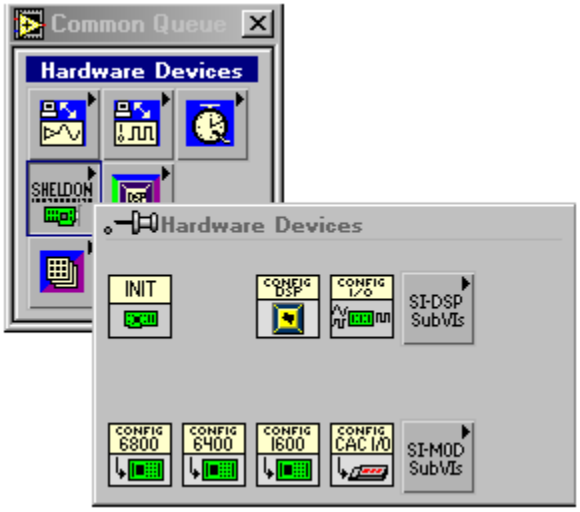
3.3.7.3 Initialize Serial Port.

Initialized the C3x's serial port, whose pins are defaulted to be used as general purpose digital IO port or it can be reconfigured to be used to implement digital counters.



3.4 Common Queue: Hardware Devices.

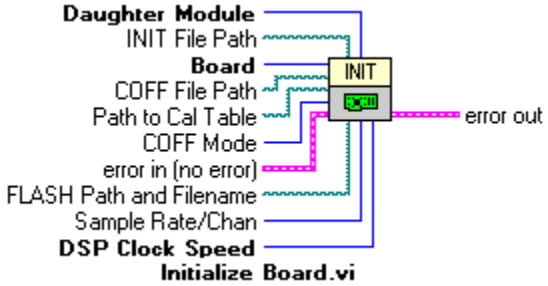
These VIs control all SI-DSP and SI-MOD hardware.



3.4.1 Initialize Board.

The Initialize Board.vi performs several tasks in the LabVIEW/host and DSP domains to prepare for a QuVIEW application.

NOTE: *The Initialize Board.vi must only be used once in the beginning of a QuVIEW application.*



INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. For all DSPs, COFF files have a *.OUT extension

and reside inside of the \SIC30DSP directory.

COFF File Path = LabVIEW filepath input used to indicate the location of the QuVIEW library COFF file for the DSP. For all DSPs, COFF files have a *.OUT extension and reside inside of the \SIC30DSP directory.

Path to Cal Table = LabVIEW filepath input used to indicate the location of the Calibration Table for the given daughter module. Please notice this table is unique to each board, and cannot be used among different boards. The values in this table reflect each analog I/O channels gain and offset errors. The calibration table file is created by the CALIBRATION.LLB, and has a *.TBL extension.

COFF Mode = LabVIEW input used to define the mode of operation of the COFF file as it relates to the Flash memory:

0 = Mode 0: Normal Load COFF Load & Run, default run mode irrespective of Flash being present or not. The COFF file is simply downloaded to the C671x's boot ROM, then the DSP is taken out of its idle or 'reset' state which triggers DSP code execution.

1 = Mode 1: Create Flash File & Run. A separate 'Flash' COFF file is created which essentially contains the QuX 'netlist' appended to the normal DSP COFF file. Once the file is created, the operation continues as in 'Mode 0' where the COFF file is simply downloaded to the C671x's boot ROM, then the DSP is taken out of its idle or 'reset' state which triggers DSP code execution.

NOTE: *One must take caution with the naming of this file, it is recomended that the '_flash' designation be appended to the end of the filename.*

2 = Mode 2: Run From Flash File. The host does nothing more than remove the DSP from its idle or 'reset' state which will force code execution.

NOTE: *The flash file must be preloaded into the Flash boot ROM in order for this mode to operate correctly.*

Flash Path and Filename = LabVIEW filepath input used to indicate the location of the Flash file. The flash file essentially contains the QuX netlist appended to the normal DSP COFF file.

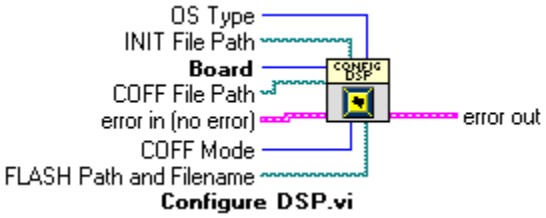
Sample Rate/Chan = LabVIEW input used to define the sample rate per channel of a multifunction analog I/O daughter module.

DSP Clock Speed = LabVIEW input to define the DSP clock speed, used to calculate various timebases.

NOTE: *The H Clock rate varies depending upon the particular DSP installed on the carrier card. Currently, all Sheldon Instruments hardware supports the following H clock rates:*
1) SI-C31DSP-PCI cards. The H clock rate for the C31 DSP ranges from 20Mhz to 30Mhz.
2) SI-C33DSP-PCI cards. The H clock rate for the C33 DSP is fixed at 75Mhz.
3) SI-C67xDSP-PCI cards. The H clock rate for the C67x DSP is fixed at 75Mhz.

3.4.2 Configure DSP.

Configures the DSP carrier card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.



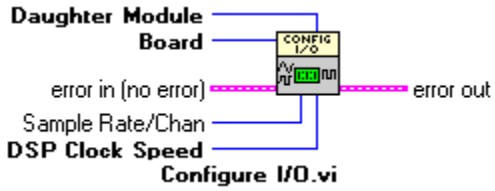
OS Type = LabVIEW input, Operating System Type. It is important to indicate what type of operating system is running, since different operating system uses different schemes for memory allocations and function calls.

INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. For C3x, it is typically C3xINI32.OUT, for the C6x it is C6711PLXINI.OUT.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

3.4.3 Configure I/O.

Configures the multifunction I/O modules that reside as daughter cards to the DSP carrier cards. The multifunction I/O cards contain ADCs, DACs, as well as digital I/O circuitry. Intended for advanced users.



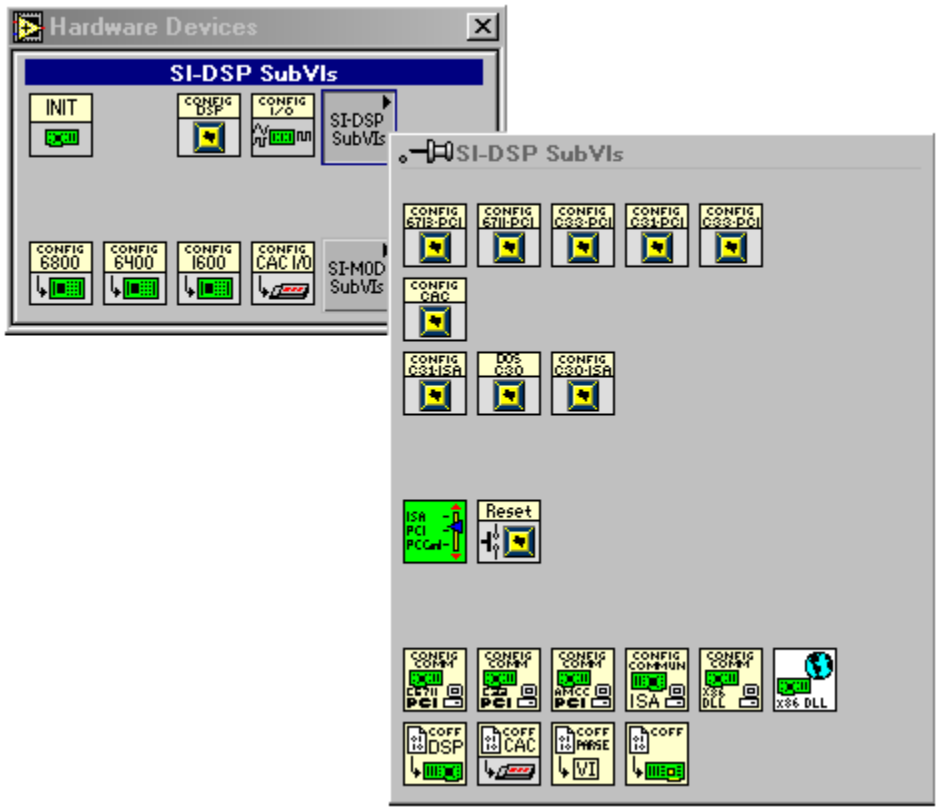
Sample Rate/Chan = LabVIEW input used to define the sample rate per channel of a multifunction analog I/O carried by the DSP board.

DSP Clock Speed = LabVIEW input to define the DSP clock speed, used to calculate sample rates for counters and timers.

NOTE: *The H Clock rate varies depending upon the particular DSP installed on the carrier card. Currently, all Sheldon Instruments hardware supports the following H clock rates:*
1) SI-C31DSP-PCI cards. The H clock rate for the C31 DSP ranges from 20Mhz to 30Mhz.
2) SI-C33DSP-PCI cards. The H clock rate for the C33 DSP is fixed at 75Mhz.
3) SI-C67xDSP-PCI cards. The H clock rate for the C67x DSP is fixed at 75Mhz.

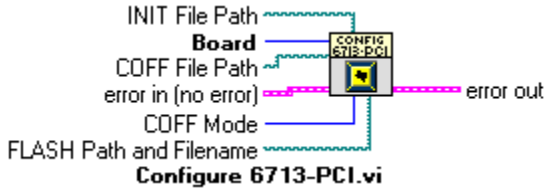
3.4.4 SI-DSP SubVIs.

These VIs are for configuring SI hardware. They are unique to specific hardware installed.



3.4.4.1 Configure C6713-PCI.

Configures the SI-C6711DSP-PCI carrier card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.



INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. For the C6711, it is C6713PLXINI.OUT.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

COFF Mode = LabVIEW input used to define the mode of operation of the COFF file as it relates to the Flash memory:

0 = Mode 0: Normal Load COFF Load & Run, default run mode

irrespective of Flash being present or not. The COFF file is simply downloaded to the C671x's boot ROM, then the DSP is taken out of its idle or 'reset' state which triggers DSP code execution.

1 = Mode 1: Create Flash File & Run. A separate 'Flash' COFF file is created which essentially contains the QuX 'netlist' appended to the normal DSP COFF file. Once the file is created, the operation continues as in 'Mode 0' where the COFF file is simply downloaded to the C671x's boot ROM, then the DSP is taken out of its idle or 'reset' state which triggers DSP code execution.

NOTE: *One must take caution with the naming of this file, it is recomended that the '_flash' designation be appended to the end of the filename.*

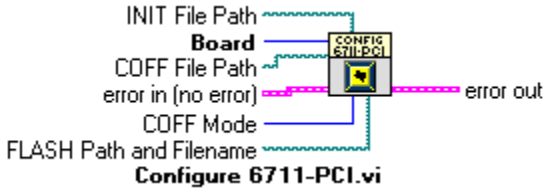
2 = Mode 2: Run From Flash File. The host does nothing more than remove the DSP from its idle or 'reset' state which will force code execution.

NOTE: *The flash file must be preloaded into the Flash boot ROM in order for this mode to operate correctly.*

Flash Path and Filename = LabVIEW filepath input used to indicate the location of the Flash file. The flash file essentially contains the QuX netlist appended to the normal DSP COFF file.

3.4.4.2 Configure C6711-PCI.

Configures the SI-C6711DSP-PCI carrier card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.



INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. For the C6711, it is C6711PLXINI.OUT.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

COFF Mode = LabVIEW input used to define the mode of operation of the COFF file as it relates to the Flash memory:

0 = Mode 0: Normal Load COFF Load & Run, default run mode irrespective of Flash being present or not. The COFF file is simply downloaded to the C671x's boot ROM, then the DSP is taken out of its idle or 'reset' state which triggers DSP code execution.

1 = Mode 1: Create Flash File & Run. A separate 'Flash' COFF file is created which essentially contains the QuX 'netlist' appended to the normal DSP COFF file. Once the file is created, the operation continues as in 'Mode 0' where the COFF file is simply downloaded to the C671x's boot ROM, then the DSP is taken out of its idle or 'reset' state which triggers DSP code execution.

NOTE: *One must take caution with the naming of this file, it is recomended that the '_flash' designation be appended to the end of the filename.*

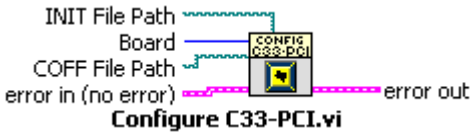
2 = Mode 2: Run From Flash File. The host does nothing more than remove the DSP from its idle or 'reset' state which will force code execution.

NOTE: *The flash file must be preloaded into the Flash boot ROM in order for this mode to operate correctly.*

Flash Path and Filename = LabVIEW filepath input used to indicate the location of the Flash file. The flash file essentially contains the QuX netlist appended to the normal DSP COFF file.

3.4.4.3 Configure C33-PCI.

Configures the SI-C33DSP-PCI carrier card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.

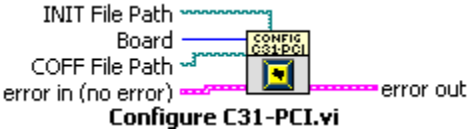


INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. For the C33, it is C33INI.OUT.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

3.4.4.4 Configure C31-PCI.

Configures the SI-C31DSP-PCI carrier card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.



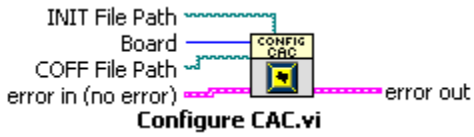
INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. For the C31, it is PCIINI32.OUT.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

3.4.4.5 Configure CAC.

Configures the SI-BULLET PCMCIA card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all

of QuVIEW's DSP resident functions. Intended for advanced users.

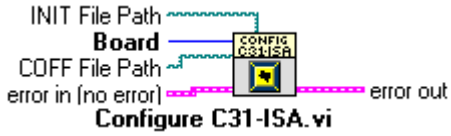


INIT File Path = Unused LabVIEW filepath input, placed for compatibility with other similar VIs.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

3.4.4.6 Configure C31-ISA.

Configures the SI-C31DSP-ISA carrier card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.

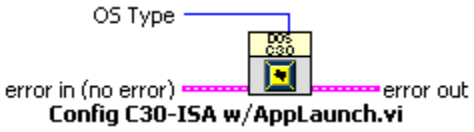


INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. For the C31, it is C31INI32.OUT.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

3.4.4.7 Configure C30-ISA w/ AppLaunch.

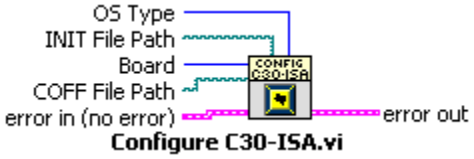
Alternate method to configure the SI-C30DSP-ISA carrier card by loading its corresponding COFF file using a command line executable. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.



OS Type = LabVIEW input, Operating System Type. It is important to indicate what type of operating system is running, since different operating system uses different schemes for memory allocations and function calls.

3.4.4.8 Configure C30-ISA.

Configures the SI-C30DSP-ISA carrier card by loading its corresponding COFF file. The COFF file contains all communications between the host PC and the DSP, along with all of QuVIEW's DSP resident functions. Intended for advanced users.



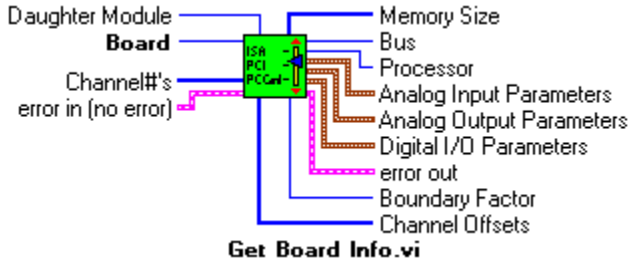
OS Type = LabVIEW input, Operating System Type. It is important to indicate what type of operating system is running, since different operating system uses different schemes for memory allocations and function calls.

INIT File Path = LabVIEW filepath input used to indicate the location of the initialization COFF file for the DSP. the the C30, it is C30INI32.OUT.

COFF File Path = LabVIEW filepath input used to indicate the location of the VQLx.OUT COFF file for the DSP.

3.4.4.9 Get Board Info.

This VI returns various useful parameters for proper hardware functionality. Intended for advanced users only.



Channel#'s = LabVIEW input array, passed on directly to Channel Offsets output array.

Memory Size = LabVIEW output array, describes the amount of memory present on each of the DSP's memory banks.

Bus = LabVIEW output that describes the bus type of the DSP card in place:

0 = ISA
1 = PCI _AMCC
2 = PCMCIA
3 = PCI _PLX
4 = Par_C6711
5 = DLL

Processor = LabVIEW output that describes the DSP processor on the carrier card:

0 = C30
1 = C31
2 = C32

3 = C33
4 = C6711
5 = X86

Analog Input Parameters = LabVIEW output cluster that defines the base address of the analog input hardware as seen by the DSP.

Analog Output Parameters = LabVIEW output cluster that defines the base address of the analog output hardware as seen by the DSP.

Digital I/O Parameters = LabVIEW output cluster that defines the base address of the digital I/O hardware as seen by the DSP.

Boundary Factor = LabVIEW output to describe the native address boundary, in terms of bytes, used by the DSP.

Channel Offsets = LabVIEW output array that directly reflects the array present on the Channels#'s input terminal.

3.4.4.10 Reset DSP.

This VI enables or disables the DSP, and only used once during card initialization. Intended for advanced users only.

Reset? = LabVIEW input which selects to Reset the DSP or not:

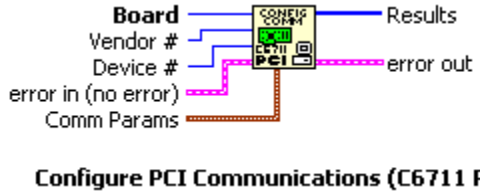
Reset?: 0 = Assert, disables the DSP.

Reset?: 1 = Deassert, activates the DSP.

Reset?: 2 = Toggle, default setting to toggles the DSP's Reset line in order to restart the COFF file loading process.

3.4.4.11 Configure PCI Communications (6711 PLX).

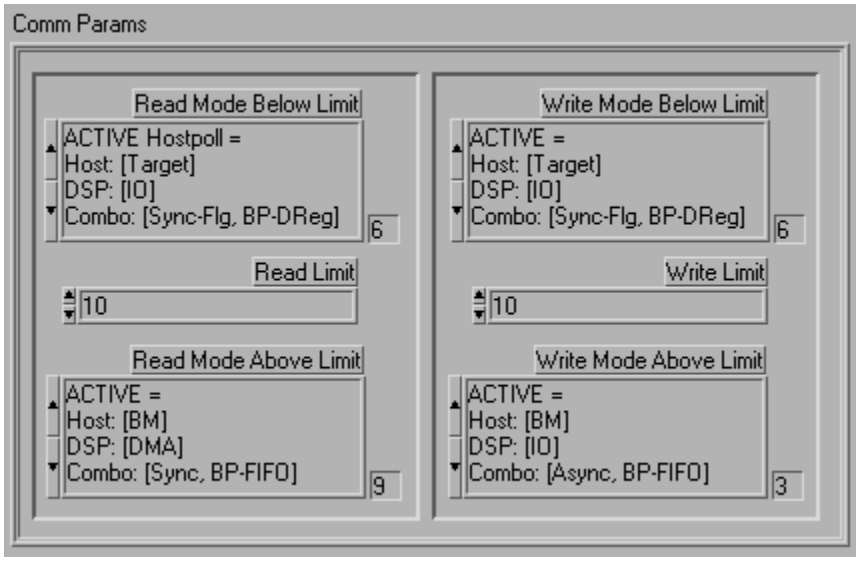
Configures the communication mode to be used between the host and the DSP for the SI-C6xDSP-PCI based boards, and only used once during card initialization.



Vendor # = Ignored.

Device # = Ignored.

Comm Params = A LabVIEW input cluster of various parameters to specify what communication mode to be used. The user can define the type of communication mode that best suits a particular application between a specific boundary. The parameters to set are as follows:



Read Mode Below Limit = Communication mode for reading below boundary limit.

Read Mode Above Limit = Communication mode for reading below boundary limit.

Read Limit = Boundary value for host reads.

Write Mode Below Limit = Communication mode for writing below boundary limit.

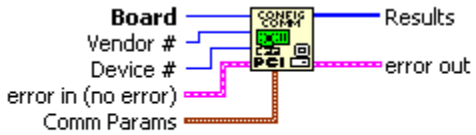
Write Mode above Limit = Communication mode for writing above boundary limit.

Write limit = Boundary value for host writes.

Results = LabVIEW output array returning the communication parameters selected.

3.4.4.12 Configure PCI Communications (C33 PLX).

Configures the communication mode to be used between the host and the DSP for the SI-C33DSP-PCI based boards, and only used once during card initialization.



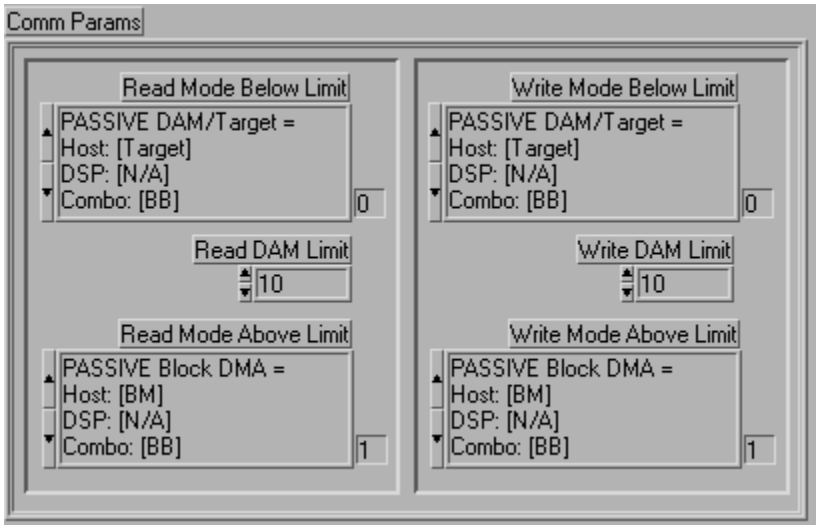
Configure PCI Communications (C33 PLX).vi

Vendor # = Ignored.

Device # = Ignored.

Comm Params = A LabVIEW input cluster of various parameters to specify what communication mode to be used. The user can define the type of communication

mode that best suits a particular application between a specific boundary. The parameters to set are as follows:



Read Mode Below Limit = Communication mode for reading below boundary limit.

Read Mode Above Limit = Communication mode for reading below boundary limit.

Read Limit = Boundary value for host reads.

Write Mode Below Limit = Communication mode for writing below boundary limit.

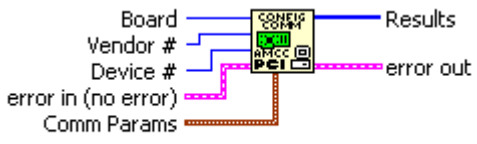
Write Mode above Limit = Communication mode for writing above boundary limit.

Write limit = Boundary value for host writes.

Results = LabVIEW output array returning the communication parameters selected.

3.4.4.13 Configure PCI Communications (AMCC).

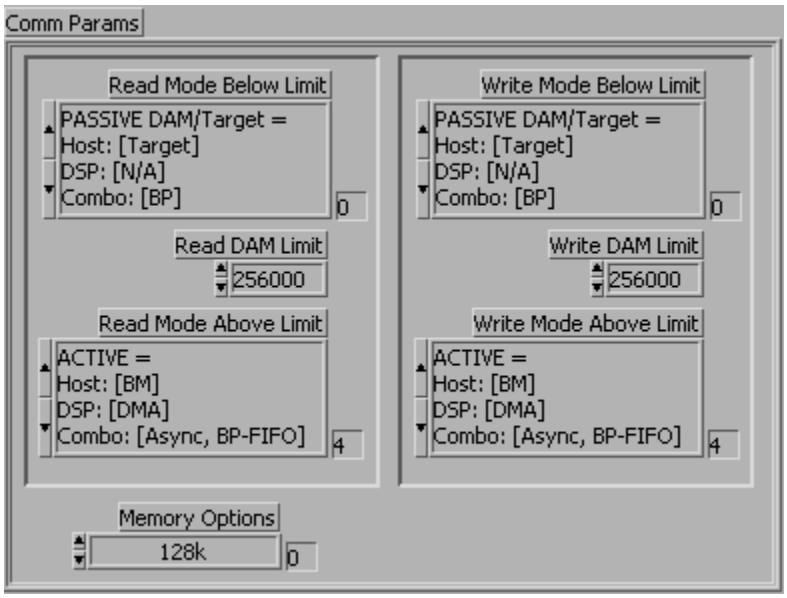
Configures the communication mode to be used between the host and the DSP for the SI-C31DSP-PCI based boards. Only executed once during the COFF file loading to the DSP.



Vendor # = Ignored.

Device # = Ignored.

Comm Params = A LabVIEW input cluster of various parameters to specify what communication mode to be used. The user can define the type of communication mode that best suits a particular application between a specific boundary. The parameters to set are as follows:



Read Mode Below Limit = Communication mode for reading below boundary limit.

Read Mode Above Limit = Communication mode for reading below boundary limit.

Read Limit = Boundary value for host reads.

Write Mode Below Limit = Communication mode for writing below boundary limit.

Write Mode above Limit = Communication mode for writing above boundary limit.

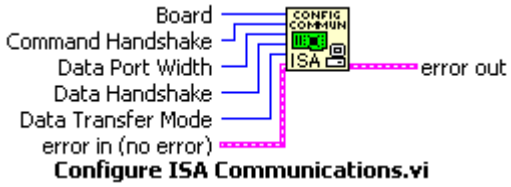
Write limit = Boundary value for host writes.

Memory options = Memory size installed on DSP carrier card.

Results = LabVIEW output array returning the communication parameters selected.

3.4.4.14 Configure ISA Communications.

Configures the communication mode to be used between the host and the DSP for the SI-C3xDSP-ISA based boards, and only used once during card initialization.



This is a low level subVI that is not typically altered by the user. However, when the IO and memory mapping is different from the software default values, these software values can be changed so as to coincide with the base addresses in hardware.

Command Handshake = LabVIEW input, handshaking protocol used only during the command phase of communications.

Data Port Width = LabVIEW input, specifies the data/communication port width present in hardware, either 16 or 32 bits wide.

Data Handshake = LabVIEW input, handshaking protocol used only during the data phase of communications.

Data Transfer Mode = LabVIEW input, specifies whether the DSP performs transfers to the host using programmed IO or it DMA engine.

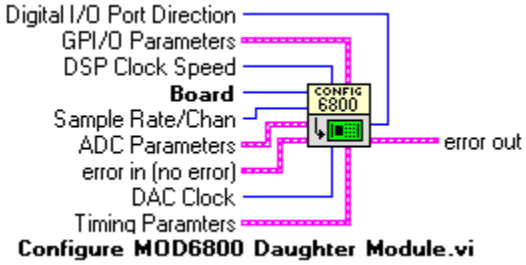
3.4.4.15 Configure DLL Communications (x86).

This VI configures communications for the host x86, used mainly for simulation purposes.

Vendor # = Ignored.
Device # = Ignored.
Results = Ignored.

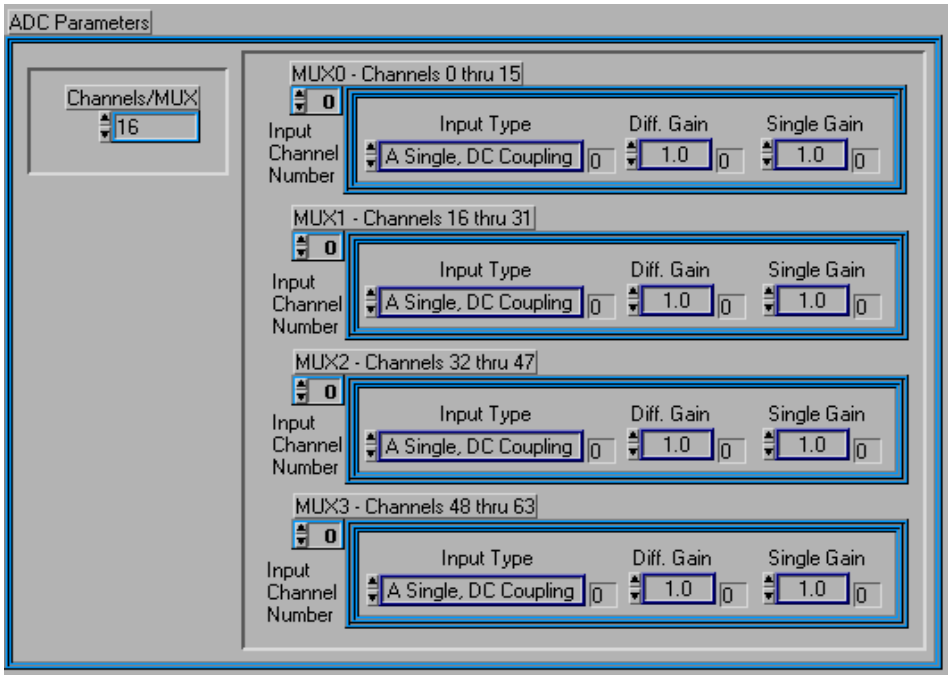
3.4.5 Configure MOD6800 Daughter Module.

Configures the SI-MOD6800 daughtermodule, please consult the SI-MOD6800 documentation for more details.

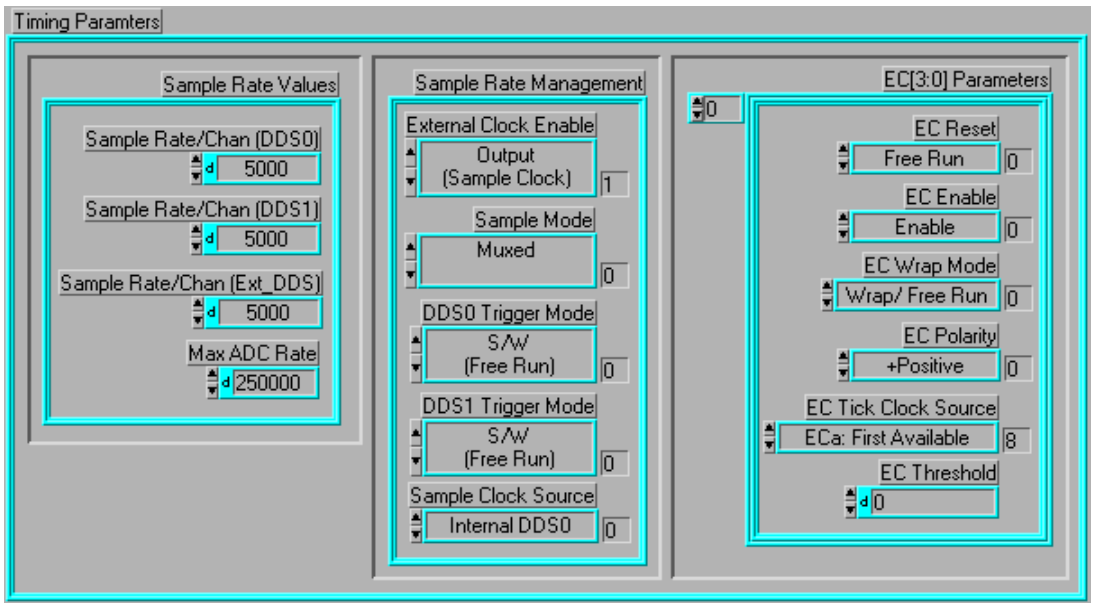


DSP Clock Speed = LabVIEW input to define the DSP clock speed.
Sample Rate/Chan = LabVIEW input used to define the sample rate per channel.
DAC Clock = LabVIEW input, not implemented.

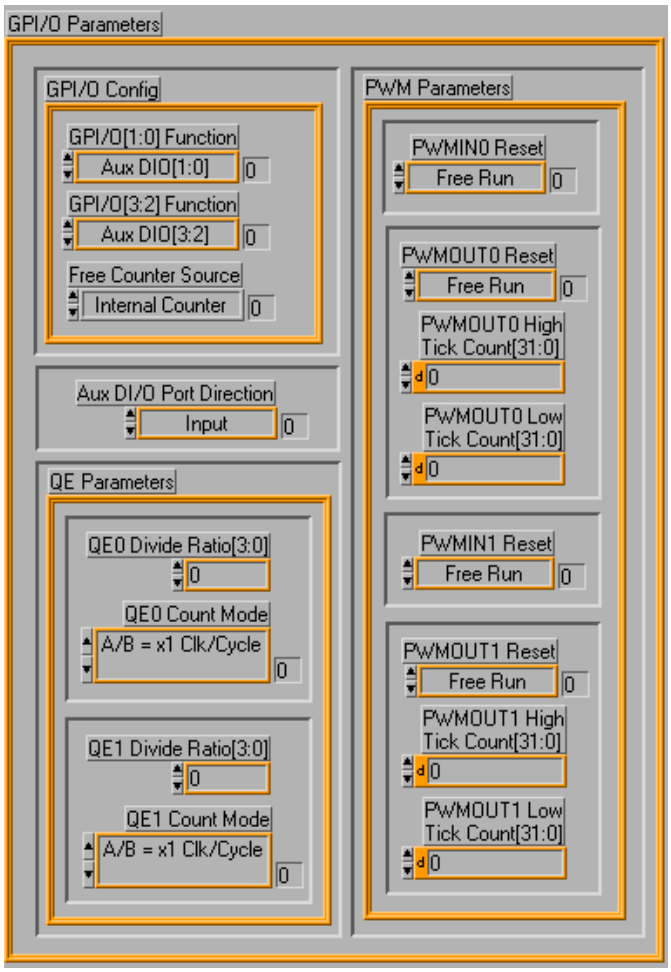
ADC Parameters = LabVIEW input cluster, defines the channel parameters as well as the number of channels to multiplex per ADC group.



Timing Parameters = LabVIEW input cluster, defines the overall timing parameters.

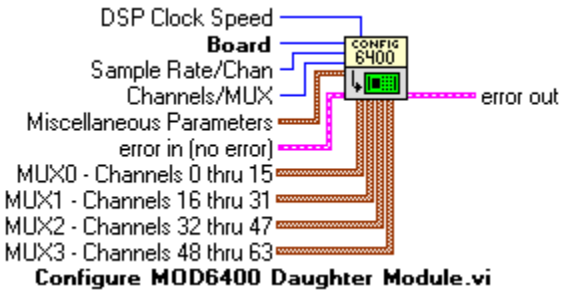


GP I/O Parameters = LabVIEW input cluster, defines the operation of the on board digital peripherals.



3.4.6 Configure MOD6400 Daughter Module.

Configures the SI-MOD6400 daughtermodule, please consult the SI-MOD6400 documentation for more details.

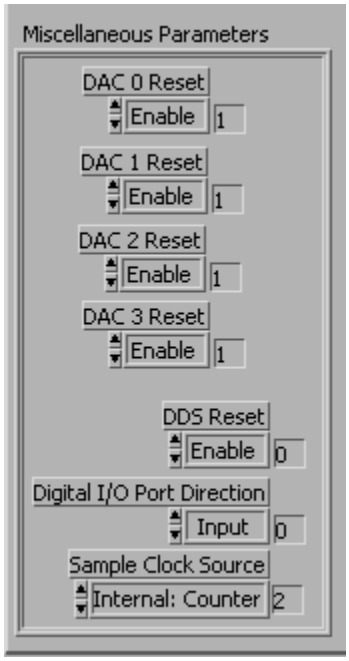


DSP Clock Speed = LabVIEW input to define the DSP clock speed.

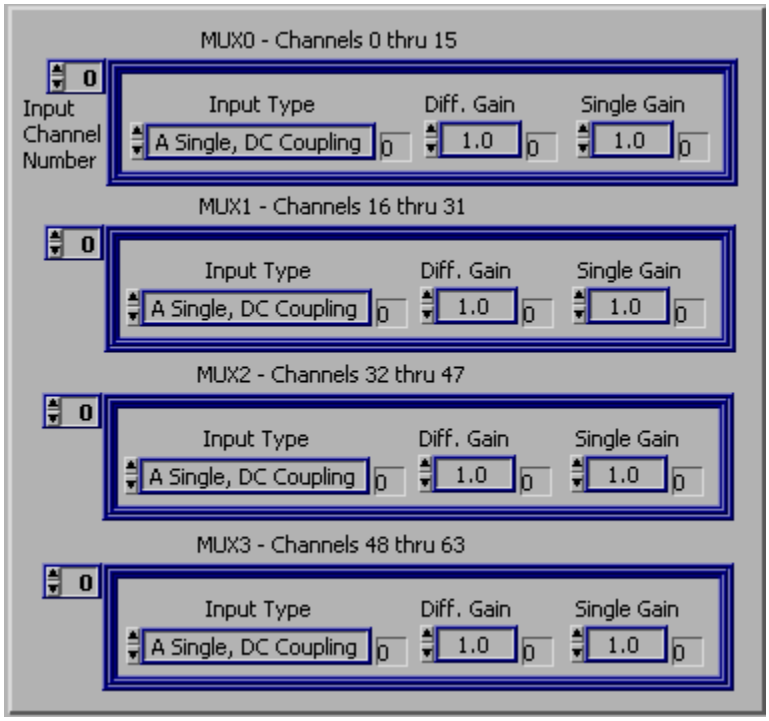
Sample Rate/Chan = LabVIEW input used to define the sample rate per channel.

Channels/MUX = LabVIEW input, defines the number of channels to multiplex per ADC group.

Miscellaneous Parameters = LabVIEW input cluster, configures miscellaneous hardware peripherals such as optional DDS and digital I/O port direction, as well as DAC enabling.

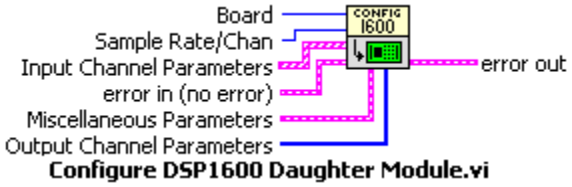


MUXn - Channels XX thru YY = LabVIEW input cluster, configures the analog input hardware operation.



3.4.6 Configure MOD1600 Daughter Module.

Configures the SI-MOD1600 daughtermodule, please consult the SI-MOD1600 documentation for more details.



Sample Rate/Chan = LabVIEW input that specifies the sample rate per channel.

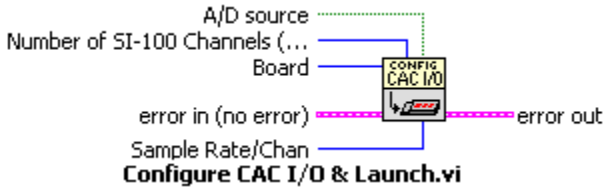
Input Channel Parameters = LabVIEW input cluster, configures the analog input hardware operation.

Output Channel Parameters = LabVIEW input array, that specifies the maximum output voltage swing to be set at either ±3Vpp or ±5Vpp.

Miscellaneous Parameters = LabVIEW input cluster, configures miscellaneous hardware peripherals such as optional DDS and DAC enabling.

3.4.7 Configure CAC I/O & Launch.

Configures the SI-Bullet I/O module, please consult the SI-Bullet documentation for more details.



AD source = LabVIEW input, determines of the analog I/O source is the SI-100 or the onboard analog I/O stereo Codecs.

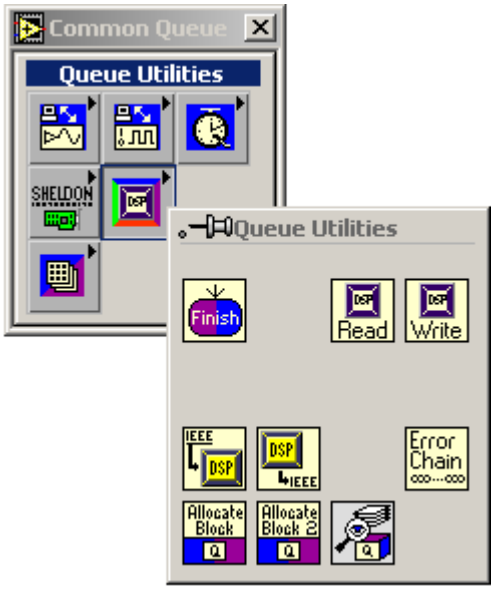
Number of SI-100 Channels = LabVIEW input, if the SI-100 is selected, specifies the number of channels transferred from the SI-100.

Sample Rate/Chan = LabVIEW input, if the onboard analog I/O stereo Codecs are selected, the sample rate per channel is specified.

3.4.8 MOD SubVIs

These are subVIs used by the SI-MOD hardware.

3.5 Common Queue: Queue Utilities.



3.5.1 Terminate Queue.

The Terminate Queue VI function is used as an indicator to the DSP that the function list is complete. After receiving this indicator, the DSP commences code execution.

Q_Input = Queuing input, may be either VQ or PQ variables, and optionally used to force presedence. The use of the 'error in' and 'error out' cluster terminals is recommended for imposing deterministic precedence.

Q_Output = Queuing output, may be either VQ or PQ variables, and optionally used to force presedence.

***NOTE**: It may be necessary to maintain precedence within LabVIEW by wiring in the Error In terminal. In this way, the Terminate Queue VI will be guaranteed to execute last. Only a single Terminate Queue.vi is required.*

3.5.2 Read from DSP board.

This VI allows the host to read one or more 32 bit words from the DSP's memory space.

Count = LabVIEW input that sets the number of sequential DWord locations that are read from the DSP.

Start Address = LabVIEW input used to indicate an address location within the DSP's address space from where the reads are to begin.

RW Mode = LabVIEW input that determines the communications mode. The following optins apply:

RW Mode: 0 = Default. Uses the mode specified in the 'Configure Communications' subVI.

RW Mode: 1 = Hostpoll. Forces the "hostpoll" mode of communications between the host and DSP; please consult the SICommModes documentation for more details.

RW Mode: 2 = Debug. Uses the actual address as seen from the host side. The Address field no longer refers to the address as seen from the DSP. Useful for debugging.

Data Read = LabVIEW array containing the requested data from the DSP.

3.5.3 Write to DSP board.

This VI allows the host to write one or more 32 bit words to the DSP's memory space.

Data to Write = LabVIEW input array containing the hex data to be written to the DSP's address space.

Start Address = LabVIEW input used to indicate an address location within the DSP's address space from where the writes are to begin.

RW Mode = LabVIEW input that determines the communications mode. The following options apply:

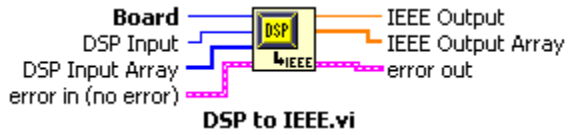
RW Mode: 0 = Default. Uses the mode specified in the 'Configure Communications' subVI.

RW Mode: 1 = Hostpoll. Forces the "hostpoll" mode of communications between the host and DSP; please consult the SICommModes documentation for more details.

RW Mode: 2 = Debug. Uses the actual address as seen from the host side. The Address field no longer refers to the address as seen from the DSP. Useful for debugging.

3.5.4 DSP to IEEE.

This VI is strictly performed on the host and serves to translate the floating point format from the DSP domain (if necessary) to the native IEEE floating point format used on the host.



DSP Input = LabVIEW input formatted with the DSP floating point scheme. Not to be used if an array terminal is wired.

DSP Input Array = LabVIEW input array formatted with the DSP floating point scheme. Convenient if an array of variables is used as it avoids the wiring of separate icons. If wired, it becomes the default terminal used, and the single variable terminal is ignored.

IEEE Output = LabVIEW output formatted with the IEEE floating point scheme for use by LabVIEW VIs. Not to be used if an array terminal was wired.

IEEE Output Array = LabVIEW output array formatted with the IEEE floating point scheme for use by LabVIEW VIs. If an input array terminal was used, then the output array should be correspondingly used.

3.5.5 IEEE to DSP.

This VI is strictly performed on the host and serves to translate the floating point format used on the host to the floating point format used on the DSP (if necessary).

IEEE Input = LabVIEW input formatted with the IEEE floating point scheme. Not to be used if an array terminal is wired.

IEEE Input Array = LabVIEW input array formatted with the IEEE floating point scheme. Convenient if an array of variables is used as it avoids the wiring of separate icons. If wired, it becomes the default terminal used, and the single variable terminal is ignored.

DSP Output = LabVIEW output formatted with the DSP floating point scheme for use by the DSP. Not to be used if an array terminal is wired.

DSP Output Array = LabVIEW output array formatted with the DSP floating point scheme for use by the DSP. If an input array terminal was used, then the output array should be correspondingly used.

3.5.6 Error Chainer.

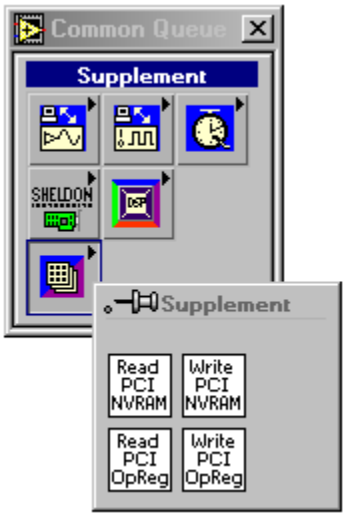
Daisy chains error codes and messages from previous VIs to the following set of VIs. If no errors ocurr, outputs simply mirror inputs.

Error = LabVIEW input, error code reference number.

Error Source = A LabVIEW string or message describing the current error source.

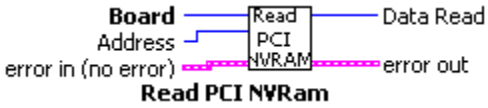
Error Out = A LabVIEW output formatted as a cluster.

3.6 Supplemental VIs.



3.6.1 Read PCI NVRAM.

Used to read bytes that reside inside of the PCI bridge device's configuration NVRAM.

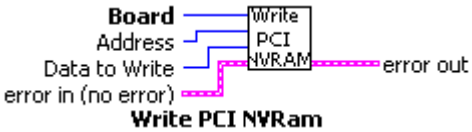


Address = LabVIEW input, byte boundary address of NVRAM.

Data Read = LabVIEW output, depending upon the PCI bridge device, either byte or DWord values are read from NVRAM.

3.6.2 Write PCI NVRAM.

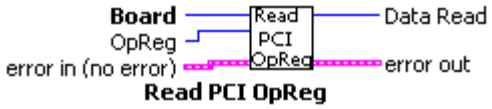
Used to write bytes to the PCI bridge device's configuration NVRAM.



Address = LabVIEW input, byte boundary address of NVRAM.
Data to Write = LabVIEW input, depending upon the PCI bridge device, either byte or DWord value is written to the NVRAM.

3.6.3 Read PCI Opreg.

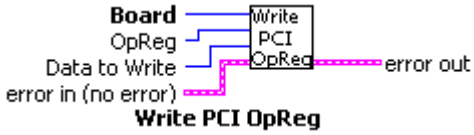
Used to read data contained in the PCI bridge device's Operation Registers.



OpReg = LabVIEW input, OpReg address in bytes.
Data Read = LabVIEW output, DWord data read from OpReg

3.6.4 Write PCI Opreg.

Used to write data to the PCI bridge device's Operation Registers.



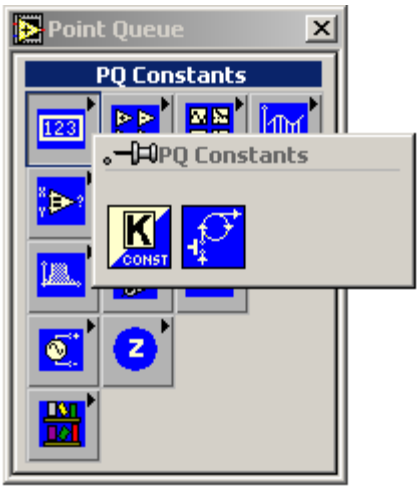
OpReg = LabVIEW input, OpReg address in bytes.

Data to Write = LabVIEW input, DWord data to write to OpReg

**$$$

4.1 PQ Constants.

These VIs are necessary because the interconnecting wires within the QuVIEW/DSP domain represent “queueing variables" instead of direct LabVIEW numbers. Therefore, the PQ_Constant is convenient for transferring LabVIEW domain numeric constants and translating them to queueing constants used within the DSP domain.



4.1.1 PQ Constant.

This VI serves to pass single precision floating point constants from the LabVIEW domain to the DSP domain. The values in the LabVIEW domain are passed onto the DSP every time it is executed by LabVIEW, thereby rendering the ability to update constants in real time to the DSP.

Constant = LabVIEW constant input. Not to be used if an array terminal is wired.

1D Array of Constants = LabVIEW input array, useful in condensing wires. Convenient if an array of variables is used as it avoids the wiring of separate icons. If wired, it becomes the default terminal used, and the single variable terminal is ignored.

PQ_Constant = PQ constant output to be used by other "PQ" inputs within the DSP domain. Not to be used if an array terminal is wired.

PQ_1D Array of Constants = PQ output array of constants, useful in condensing wires. If an input array terminal was used, then the output array should be correspondingly used.

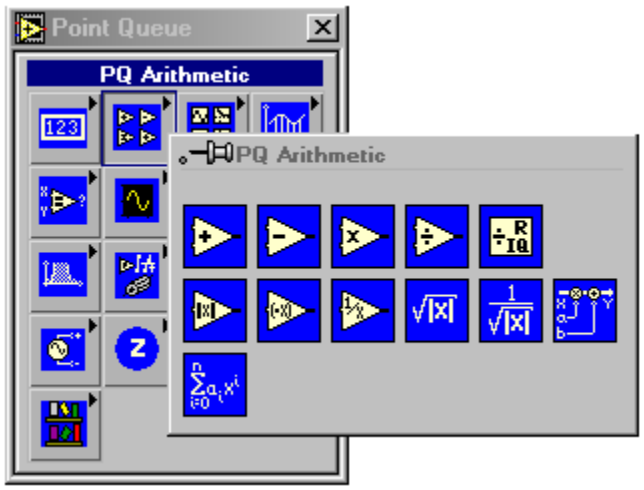
4.1.2 PQ Latch.

The PQ_Latch function allows for a PQ variable to be passed on to its output terminal as long as it is enabled to do so. If disabled, the PQ output variable is latched and remains the same, ignoring the PQ input variable.

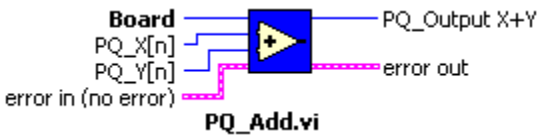
PQ_X[n] = PQ input variable to be passed on to the output terminal.
PQ_Latch Enable (Reset) = PQ input that determines if the value at the PQ_Y [n] output is a latched value or a real time value from PQ_X[n] input.
0 = Passive Output: The PQ_Y[n] output will be a direct reflection and follow the input.
1 = Latch Input: latches the PQ_Y[n] output and ignores the input.

PQ_Y[n] = PQ output.

4.2 PQ Arithmetic.



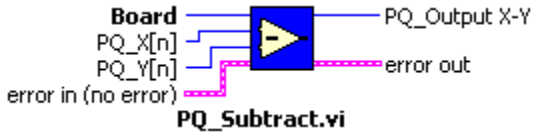
4.2.1 PQ_Add.



PQ_X[n] = PQ input.
PQ_Y[n] = PQ input.

PQ_Output X+Y = PQ output, sum of X[n] and Y[n].

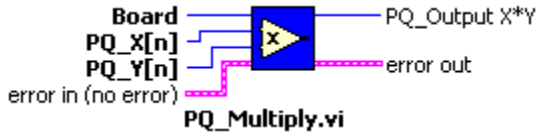
4.2.2 PQ_Subtract.



PQ_X[n] = PQ input.
PQ_Y[n] = PQ input.

PQ_Output X-Y = PQ output, difference of X[n] and Y[n].

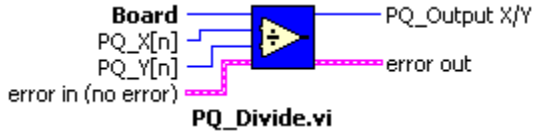
4.2.3 PQ_Multiply.



PQ_X[n] = PQ input.
PQ_Y[n] = PQ input.

PQ_Output X*Y = PQ output, product of X[n] and Y[n].

4.2.4 PQ_Divide.



PQ_X[n] = PQ input.
PQ_Y[n] = PQ input.

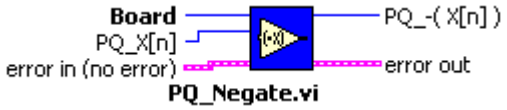
PQ_Output X/Y = PQ output, ratio of X[n] over Y[n].

4.2.5 PQ_Absolute Value.

PQ_X[n] = PQ input.

PQ_|X[n]| = PQ output, absolute value of X[n].

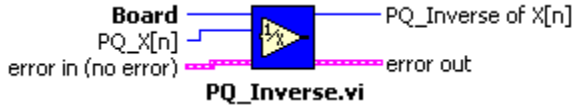
4.2.6 PQ_Negate.



PQ_X[n] = PQ input.

PQ_-(X[n]) = PQ output, negative of X[n].

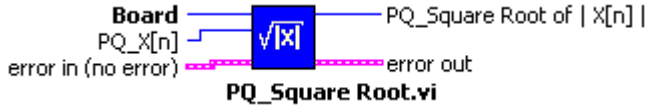
4.2.7 PQ_Inverse.



PQ_X[n] = PQ input.

PQ_Inverse of X[n] = PQ output, inverse of X[n].

4.2.8 PQ_Square Root.



PQ_X[n] = PQ input.

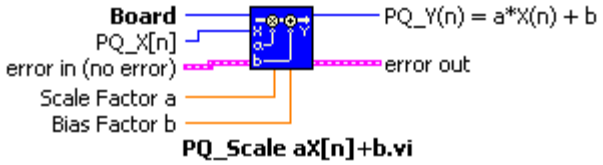
PQ_Square Root of |X[n]| = PQ output, square root of absolute value of X[n].

4.2.9 PQ_Inverse Square Root.

PQ_X[n] = PQ input.

PQ_Inverse Square Root of |X[n]| = PQ output, inverse square root of absolute value of X[n].

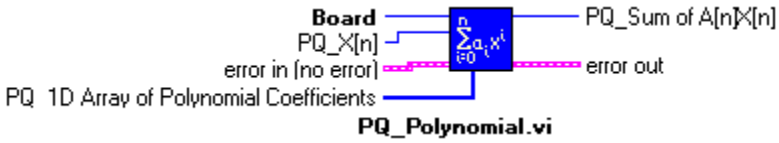
4.2.10 PQ_Scale aX+b.



PQ_X[n] = PQ input.
Scale Factor a = LabVIEW constant that multiplies X[n].
Bias Factor b = LabVIEW constant that is added to the product of a*X[n].

PQ_Scale a*X[n]+b = PQ output, single step procedure for scaling and biasing X [n].

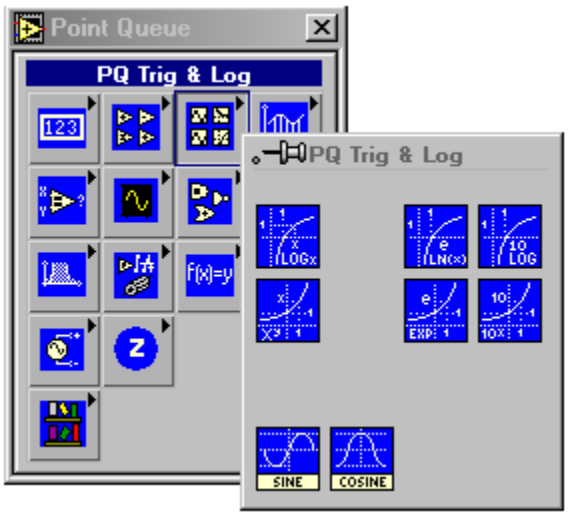
4.2.11 PQ_Polynomial.



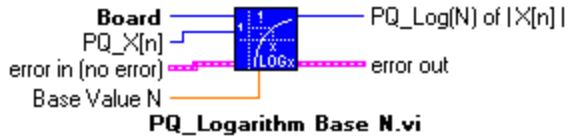
PQ_X[n] = PQ input.
PQ_1D Array of Polynomial Coefficients = PQ 1D Array input, where the nth position of the coefficient in the array corresponds to the nth power term.

PQ_Sum of A[n]X[n] = PQ output, the result of the sumation of all the polynomial terms.

4.3 PQ Trig and Log.



4.3.1 PQ_Logarithm Base N.



PQ_X[n] = PQ input.
Base Value N = LabVIEW input, the base value for which to compute the Logarithm.

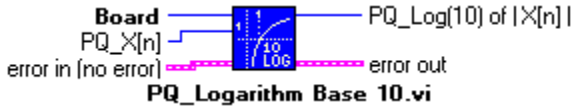
PQ_Log(N) of |X[n]| = PQ output, computes the logarithm base "N" with argument as the absolute value of X[n].

4.3.2 PQ_Natural Logarithm.

PQ_X[n] = PQ input.

PQ_Natural Log of |X[n]| = PQ output, computes natural logarithm base "e" with argument as the absolute value of X[n].

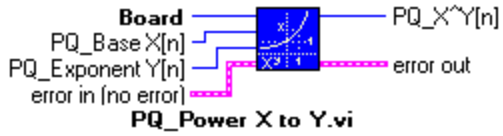
4.3.3 PQ_Logarithm Base 10.



PQ_X[n] = PQ input.

PQ_Log(10) of |X[n]| = PQ output, computes the logarithm base "10" with argument as the absolute value of X[n].

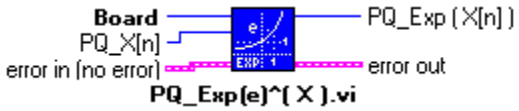
4.3.4 PQ_Power X to Y.



PQ_Base X[n] = PQ input, base.
PQ_Exponent Y[n] = PQ input, exponent.

PQ_X^Y[n] = PQ output, computes the exponent of base "X[n]" to the power of Y[n].

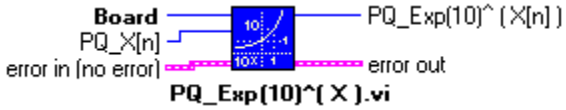
4.3.5 PQ_Exp(e)^(X).



PQ_X[n] = PQ input.

PQ_Exp(X[n]) = PQ output, computes exponential base "e" with power of X[n].

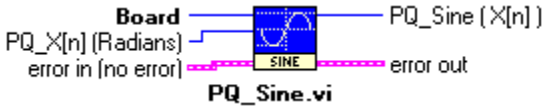
4.3.6 PQ_Exp(10)^(X).



PQ_X[n] = PQ input.

PQ_Exp(10)(X[n]) = PQ output, computes exponential base "10" with power of X[n].

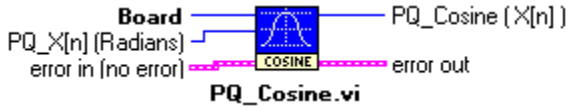
4.3.7 PQ_Sine.



PQ_X[n] (Radians) = PQ input, represented as radians.

PQ_Sine(X[n]) = PQ output, computes sine of radians of X[n].

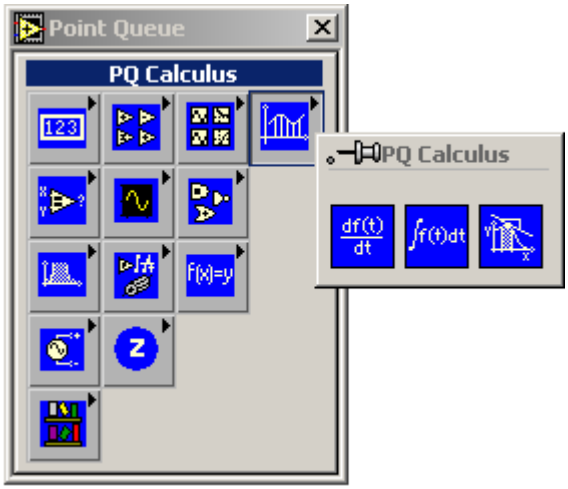
4.3.8 PQ_Cosine.



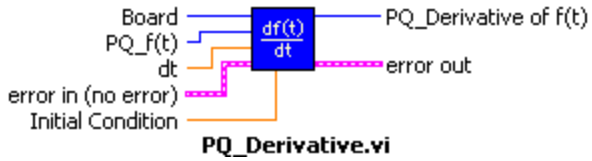
PQ_X[n] (Radians) = PQ input, represented as radians.

PQ_Cosine(X[n]) = PQ output, computes cosine of radians of X[n].

4.4 PQ Calculus.



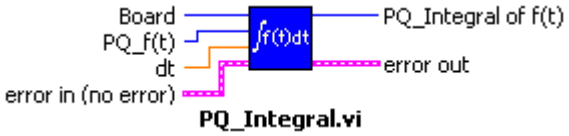
4.4.1 PQ_Derivative.



PQ_f(t) = PQ input.
dt = LabVIEW input that determines width at which to divide the current and previous values of f(t).

PQ_Derivative of f(t) = PQ output that returns a continuous differentiating output.

4.4.2 PQ_Integral.



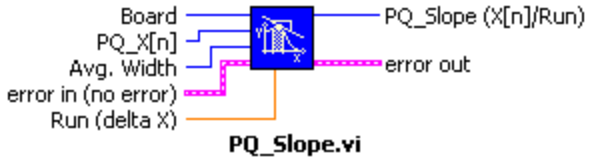
PQ_f(t) = PQ input.

PQ_Integral of f(t) = PQ output that returns a continuos additive output.

NOTE: It is possible to observe a wrap around effect (also known as wind up) when using the Integration function. This is not an error in the DSP code, but rather the compounded effect of the input signal's offset added back onto itself. The Calibrate function within the miscellaneous operations subset may be used to digitally correct this initial input signal offset.

4.4.3 PQ_Slope.

This VI computes the slope of a sequence of incoming samples.



PQ_X[n] = PQ input.

Avg. Width = LabVIEW input that defines the number of points to be used to compute the slope.

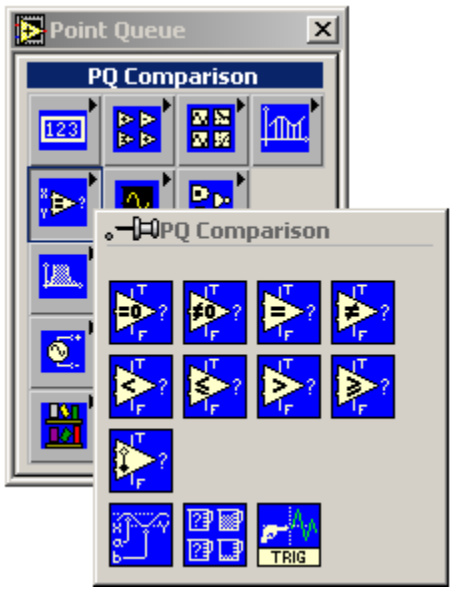
Run (delta X) = LabVIEW input that defines each abscissa increment or the difference in the X axis between subsequent points.

PQ_Slope (X[n]/Run) = PQ Output, slope obtained by dividing X[n]/Run

---

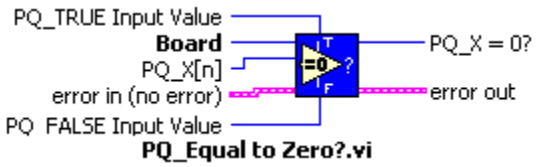
4.5 PQ Comparison.

The 'Q_Equal?', 'Q_Not equal?', 'Q_Less Than', 'Q_Less Than or Equal?', 'Q_Greater Than?', 'Q_Greater Than or Equal?' functions take in a pair of queueing inputs that will be subjected to the comparison, along with a pair of queueing inputs that define the boolean TRUE/FALSE values to be returned as the result. In other words, the output returns the value wired to the "TRUE" input if the case is a true, or the output returns the value wired to the "FALSE" input if the case is false. Each TRUE or FALSE state is defined as a pair of queuing inputs; i.e. a TRUE state can be defined as one value and a FALSE state as another value. Either state definition can be derived from any queuing function.



4.5.1 PQ_Equal to Zero?

Compares the PQ input X[n] to PQ "zero" value and returns a "PQ True" if equal, and a "PQ False" if not.

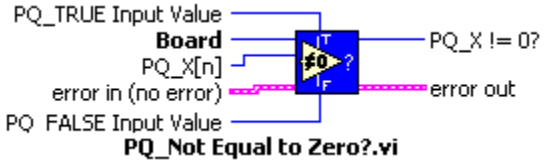


PQ_X[n] = PQ Input value to be compared to a PQ zero.

PQ_X = 0? = PQ output, PQ TRUE/FALSE result of the comparison.

4.5.2 PQ_Not Equal to Zero?

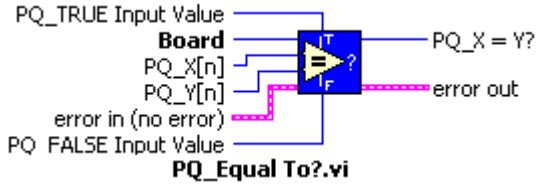
Compares the PQ input X[n] to a PQ "zero" and returns a "PQ False" if equal, and a "PQ True" if NOT equal.



PQ_X[n] = PQ Input value to be compared to a PQ non-zero value.

PQ_X = 0? = PQ output, PQ TRUE/FALSE result of the comparison.

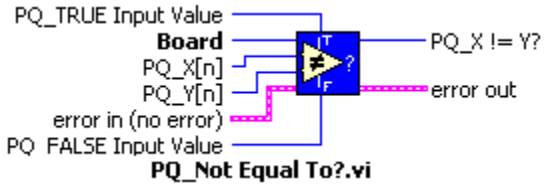
4.5.3 PQ_Equal?



PQ_X[n] = First PQ input to compare.
PQ_Y[n] = Second PQ input to compare.

PQ_X=Y? = PQ output that returns the value wired to the PQ_TRUE if X=Y, or the value wired to the PQ_FALSE if X!=Y.

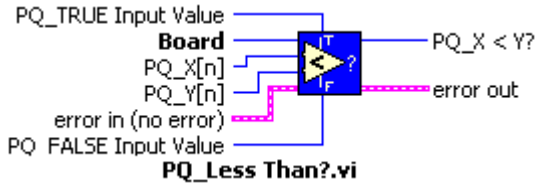
4.5.4 PQ_Not Equal?



PQ_X[n] = First PQ input to compare.
PQ_Y[n] = Second PQ input to compare.

PQ_X != Y? = PQ output that returns the value wired to the PQ_TRUE if X!=Y, or the value wired to the PQ_FALSE if X=Y.

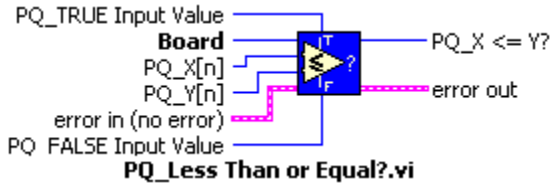
4.5.5 PQ_Less Than?



PQ_X[n] = First PQ input to compare.
PQ_Y[n] = Second PQ input to compare.

PQ_X<Y? = PQ output that returns the value wired to the PQ_TRUE if X<Y, or the value wired to the PQ_FALSE if X >= Y.

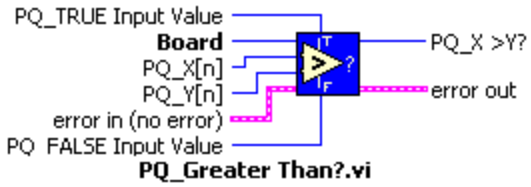
4.5.6 PQ_Less Than or Equal?



PQ_X[n] = First PQ input to compare.
PQ_Y[n] = Second PQ input to compare.

PQ_X<=Y? = PQ output that returns the value wired to the PQ_TRUE if X<=Y, or the value wired to the PQ_FALSE if X>Y.

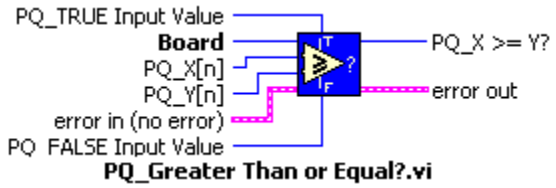
4.5.7 PQ_Greater Than?



PQ_X[n] = First PQ input to compare.
PQ_Y[n] = Second PQ input to compare.

PQ_X>Y? = PQ output that returns the value wired to the PQ_TRUE if X>Y, or the value wired to the PQ_FALSE if X<=Y.

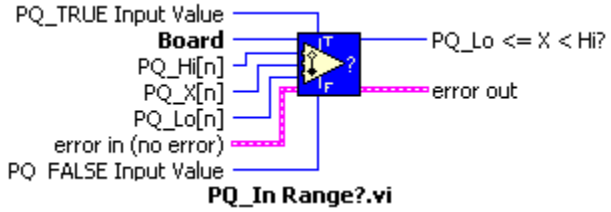
4.5.8 PQ_Greater Than or Equal?



PQ_X[n] = First PQ input to compare.
PQ_Y[n] = Second PQ input to compare.

PQ_X>=Y? = PQ output that returns the value wired to the PQ_TRUE if X>=Y, or the value wired to the PQ_FALSE if X<Y.

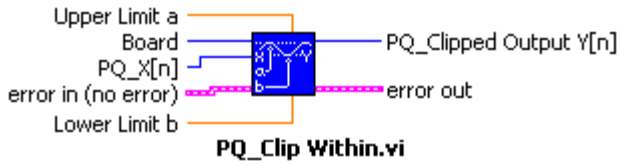
4.5.9 PQ_In Range?



PQ_Hi[n] = PQ input, upper limit value.
PQ_X[n] = PQ input to compare.
PQ_Lo[n] = PQ input, lower limit value.

PQ_Lo <= X < Hi? = PQ output that returns the value wired to the PQ_TRUE if X falls within the two limits, or the value wired to the PQ_FALSE if X does not.

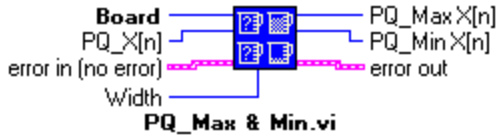
4.5.10 PQ_Clip Within.



PQ_X[n] = PQ input.
Upper Limit a = LabVIEW constant used to determine the upper limit to clip X [n].
Lower Limit b = LabVIEW constant used to determine the lower limit to clip X [n].

PQ_Clipped Output Y[n] = PQ output reflecting the input X[n] bound by the upper and lower limits.

4.5.11 PQ_Max & Min.

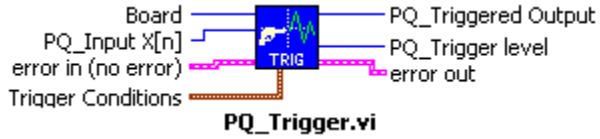


PQ_X[n] = PQ input.
Width N = LabVIEW constant to determine the number of points inside a moving block from which to extract the high and low peak values.

PQ_Max X[n] = PQ output of the maximum value from a block of "N" points.
PQ_Min X[n] = PQ output of the minimum value from a block of "N" points.

4.5.12 PQ_Trigger.

The PQ_Trigger function is the classical implementation of an analog trigger detector. When the trigger conditions are met, a PQ "zero" value or TRUE is returned, while a PQ non-zero value indicates that the conditions are not met. The resultant of the trigger function may can be used to synchronize any signal to any time dependent event. It can be fed into the "PQ_Sync In" or "PQ_Trig_In" inputs of the PQ_Index, PQ_Peak Detection, PQ_Scope, PQ_Packed Scope, and PQ_Max & Min functions.



PQ_X[n] = PQ input signal to be examined.
Trigger Conditions = LabVIEW cluster used to determine the conditions upon which a trigger condition is met. Please refer to the next section for more details.

PQ_Triggered Output = PQ output that returns a low asserted value or a PQ "zero" for a true or trigger condition met, or a high value or PQ non-zero value for a false or no trigger.

PQ_Trigger level = PQ output value representing the value of the input when the trigger condition ocurred.

4.5.12.1 Trigger Conditions.

1) Trigger Level: The TRIGLEVEL value determines the voltage level at which the specified trigger channel must intersect. The trigger voltage is specified within the maximum range of the analog input of one of the daughter modules. Triggering is ignored if the slope is disabled or set to none.

2) Trigger Slope: The TRIGGER SLOPE setting determines the direction or slope at which the specified trigger channel must cross the trigger level. Triggering can be set on a positive slope, negative slope, or be disabled if set to none.

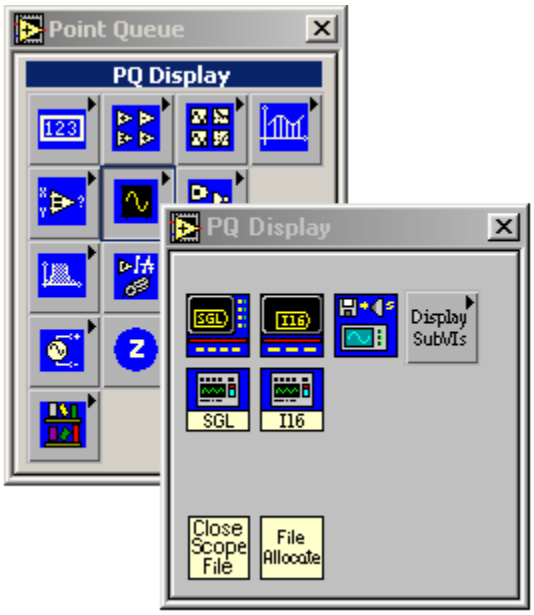
3) Hysteresis: The HYSTERESIS value determines a voltage window which the trigger channel must exceed before intersecting the trigger level. The hysteresis is an absolute value, whereby false triggers can be avoided.

For example, a hysteresis value of 1V means the signal must drop 1V below the trigger level, in order for a trigger to occur on a positive slope; or the signal must rise 1V above the trigger level for the negative slope case.

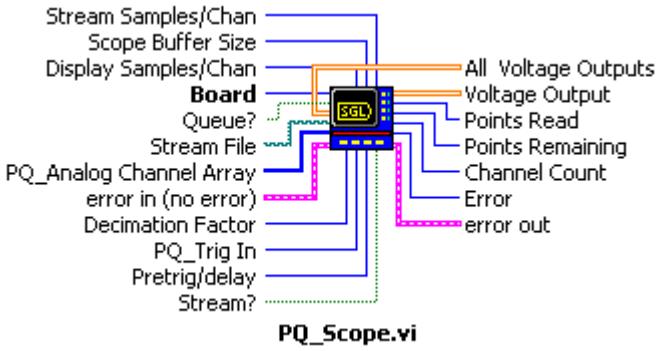
4) Pretrigger/Delay: The PRETRIG/DELAY specifies the number of samples to keep track of before or after a trigger event. A negative entry for Pretrig/Delay specifies the number of data points to keep track of before a trigger event, thereby defining a starting point to collect and display any data specified in the Samples entry. A "0" entry or a positive entry for Pretrig/Delay specifies the number of data points to wait after a trigger event before collecting and displaying any data specified in the Samples entry.

Note: The Pretrigger/Delay is variable is not defined within the PQ_Trigger function, but rather in the PQ_Scope functions. Please refer to the next section for more details.

4.6 PQ Display & Log to Disk.



4.6.1 PQ_Scope.



Queue? = LabVIEW boolean input that specifies if the VI is to be executed in a queuing sequence or not.

Stream File = LabVIEW filepath input that specifies the target file to record data during stream to disk operations.

PQ_Analog Channel Array = PQ input array that specifies the PQ input sources to be displayed or recorded to disk.

NOTE: *The format of the PQ data wired to the PQ_Analog Channel Array terminal may represent scaled or floating point values only.*

Display Samples/Chan = LabVIEW input that specifies the number of points to be displayed per channel.

Scope Buffer Size = LabVIEW input that specifies the maximum buffer set aside within the DSP's memory space. A recommended size of 32k is appropriate for most applications, when more SRAM memory is available, increasing the buffer size to 64k will improve data transfer speeds.

Stream Samples/Chan = LabVIEW input that specifies the number of points to be recorded to disk.

Decimation Factor = LabVIEW input used to specify the number of points to drop for display or disk recording.

PQ_Trig In = PQ input that specifies a PQ trigger source.

Pretrig/Delay = LabVIEW input that takes in a negative value for pretriggering and positive values for postriggering, this value is limited by the amount allotted in Scope Buffer Size input.

Stream? = LabVIEW boolean input that specifies if the outgoing data is being written to a target file or not.

Voltage Output = LabVIEW two dimensional output array, that contains the voltage scaled data to be used by LabVIEW.

All Voltage Outputs = LabVIEW one dimensional output array, very similar to Voltage Output, except all channel data is interleaved.

Points Read = LabVIEW output that indicates the number of points read through a queuing iteration.

Points Remaining = LabVIEW output that indicates the number of points remaining during disk recording.

Channel Count = LabVIEW output that indicates the number of channels used for display or disk recording.

The PQ_Scope VI allows LabVIEW to receive data from the DSP. This data can then be displayed and/or streamed to disk. The minimal inputs to the PQ_Scope VI, whether streaming or displaying, is an array wired into the PQ_Analog Channel Array. This array can be built by wiring in any PQ variable within the block diagram. The data is returned to LabVIEW in the form of a 32 bit floating point voltage formatted as a two dimensional array. With PQ_Analog Channel Array being the only input wired, PQ_Scope defaults to a display of 256 points per channel. The input to Display Samples/chan can be wired to any desired value.

NOTE: *Unlike the rest of the queuing functions, the PQ_Scope function can only be used once in a queuing sequence. However, a duplicate may be made in a sequence that follows the PQ_Finish subVI with the 'Queuing?' input wired to a 'false', in order to be used in conjunction with LabVIEW function icons. Conversely, if a second icon is not used, its output array may passed to a later sequence by using either a LabVIEW Sequence Local or a LabVIEW Local Variable.*

The PQ_Scope VI can also stream data to disk. To do so, a File Path must be wired into Stream File and a LabVIEW value must be wired to the Stream Samples/chan input. Lastly, to set PQ_Scope in stream mode, one must wire the Stream? boolean input to a LabVIEW True, since by default this input is set to a LabVIEW False.

The PQ_Scope VI can simultaneously display and stream to disk by simply wiring each appropriate input for both display and stream as previously outlined. The display may slow the streamer down enough that erroneous data may be logged to disk. If this occurs, PQ_Scope VI will return an overflow error. This may be remedied by the following:

1) Reducing the number of samples or channels.
2) Closing all subVIs during execution.
3) Closing any Windows applications which may be stealing time.
4) Changing graph attributes such as size, autoscaling, smooth updates, etc.

If overflow errors continue, disconnect the output graph and try to stream without simultaneously running the display.

The PQ_Trigger VI can be used to trigger either the streamer or display by wiring its output to the PQ_Trig In input. The trigger functions are the same as described in a previous section covering dedicated Oscilloscope VIs. While in display mode, each display buffer will be based on the trigger event. If streaming, the trigger will only occur during the first buffer. The Pretrig/delay input can be wired if desired. This input can be changed during execution to slide the trigger event while displaying data.

Besides the two dimensional array outputs, other outputs include; an indicator showing number of points left/channel; another show the number of points in the current two dimensional array/channel; as well as an error indicator with the following codes:

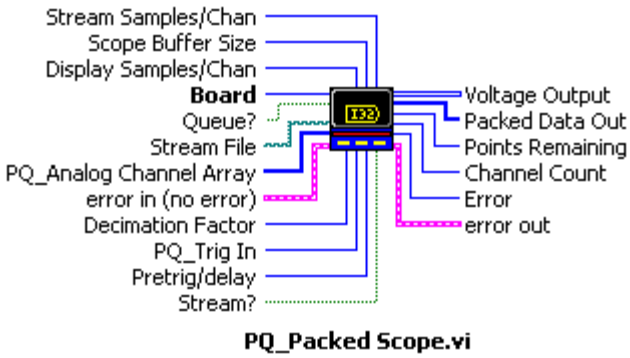
0) No Error
1) Underflow
2) Trigger Conditions not Met
3) board disconnected
-1) Overflow

The decimation input allows for only a portion of the acquired data to be passed over to the host display or disk.

Decimation has the effect of the acquired data dropping every nth point. Therefore, a decimation factor of "0" will not drop points, but a "1" will drop every other point. This will reduce your effective stream rate by a factor of 1/(n+1). Additionally, excessive underflows may occur during the scope mode of operation. During an underflow, the last valid buffer will be redisplayed. Reducing the number of display points will account for a more real-time acquisition.

The PQ_Scope VI can also be executed outside of the queuing sequence. To do so, only the PQ_Analog Channel Array input must be wired within the queuing sequence, and no graphs should be connected to the output arrays. As a result, another instance of the PQ_Scope VI can be implemented with a different VI altogether. Be sure to include the same inputs to the PQ_Analog Channel Array as was wired in the original queuing sequence; additionally, wire the boolean Queue? input to a LabVIEW False.

4.6.2 PQ_Packed Scope.



It acts in most respects like the PQ_Scope, except that data is transferred from the DSP as two "packed" 16-bit integers, where two 16 bit integer words are concatenated to form a single 32 bit word.

Queue? = LabVIEW boolean input that specifies if the VI is to be executed in a queuing sequence or not.

Stream File = LabVIEW filepath input that specifies the target file to record data during stream to disk operations.

PQ_Analog Channel Array = PQ input array that specifies the PQ input sources to be displayed or recorded to disk.

***NOTE**: The format of the PQ data wired to the PQ_Analog Channel Array terminal cannot represent scaled or floating point values, but rather integer values not exceeding 16 bits in size. Therefore, only the values returned by the 'Unscaled Analog Input' VIs may be used.*

Display Samples/Chan = LabVIEW input that specifies the number of points to be displayed per channel.

Scope Buffer Size = LabVIEW input that specifies the maximum buffer set aside during display or disk recording. The physical amount of SRAM memory shipped with the DSP board is the only limiting factor, where if shipped with the default 64kW, a recommended size of 16,000 is appropriate. When more SRAM memory is available, a larger buffer of approximately 32,000 would render faster data transfers.

Stream Samples/Chan = LabVIEW input that specifies the number of points to be recorded to disk.

Decimation Factor = LabVIEW input used to specify the number of points to drop for
display or disk recording.

PQ_Trig In = PQ input that specifies a PQ trigger source.

Pretrig/Delay = LabVIEW input that takes in a negative value for pretriggering and positive values for postriggering this value is limited by the amount allotted in Scope Buffer Size input.

Stream? = LabVIEW boolean input that specifies if the outgoing data is being written to a target file or not.

Voltage Output = LabVIEW 2D output array that contains the scaled voltage data.

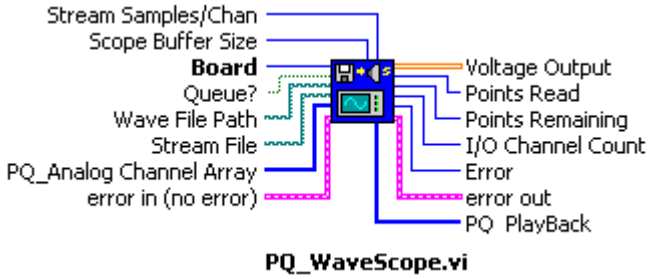
Packed Data Out = LabVIEW 1D output array of 16 bit integers, with all channel data interleaved.

Points Read = LabVIEW output that indicates the number of points read through a queuing iteration.

Points Remaining = LabVIEW output that indicates the number of points remaining during disk recording.

Channel Count = LabVIEW output that indicates the number of channels used for display or disk recording.

4.6.3 PQ_WaveScope.



The PQ_WaveSope VI is used for simultaneous stream to disk and playback from disk. This same functionality can be achieved using several discrete VIs. However, for convenience this special purpose VI will allow both operations to be performed more efficiently.

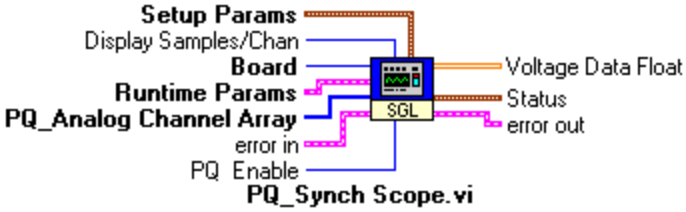
The inputs to this VI are analogous to those used on the PQ_Scope VIs reviewed above.

***NOTE**: The number of channels to record must be equal to the number of channels to playback.*

PQ_Playback = PQ 1D output array containing the playback data.

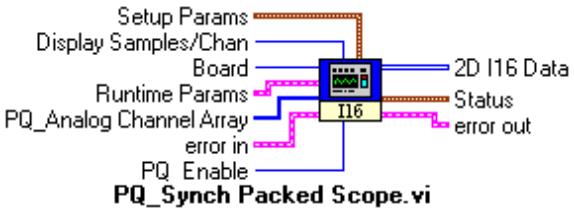
WaveFilePath = Path and filename of the playback file.
StreamFile = Path and filename of the file to be recorded.

4.6.4 PQ_Synch Scope.vi



This VI synchronously transfers data from a DSP buffer to host memory. The data may be displayed on the screen as an oscilloscope as well as be transferred to disk. Each data point transferred is represented as a 32 bit, single precision floating point value.

4.6.5 PQ_Synch Packed Scope.vi

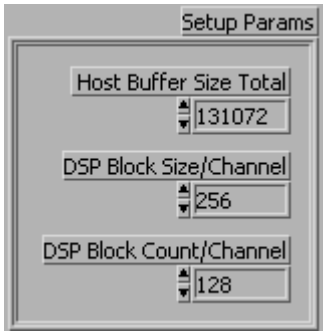


This VI synchronously transfers data from a DSP buffer to host memory. The data may be displayed on the screen as an oscilloscope as well as be transferred to disk. Each data point transferred is represented as two 16 bit integers, concatenated to form a single 32 bit fixed point value. Concatenating or packing two 16 bit words into a single 32 bit word effectively increases data transfer bandwidth, as well as doubling the amount of data points within a given block of requested memory in both the DSP and the host domains.

4.6.6 Common Synchronous Scope Parameters.

Setup Parameters = LabVIEW cluster representing parameters to configure the operation of the VI; they include.

Note: These parameters are to be set before running the application VI, and cannot be altered during runtime or after the DSP is active running an application.



Host Buffer Size Total = LabVIEW input that determines the host's total buffer size to be allocated to support the appropriate amount of data to be transferred. It is recommended to be at least 1.5 times larger than the 'Display Samples/Chan' variable.

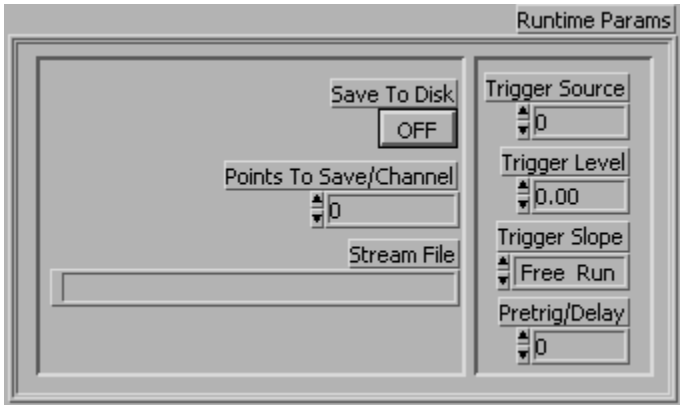
DSP Block Size/Channel = LabVIEW input that determines the number of points per channel contained in each block to be transferred fro the DSP. Each time the host reads points from DSP, it is implemented by block multiples.

DSP Block Count/Channel = LabVIEW input that determines the number of blocks to be allocated within the DSP's memory space; typcially set between 32K to 64K.

*Note: The total amount of memory allocated within DSP calculated as (DSP Block Size/Channel) * (DSP Block Count/Channel) * (Number of Channels).*

Display Samples/Chan = LabVIEW input that specifies the number of points to be displayed per channel.

Runtime Parameters = LabVIEW input cluster representing parameters that may be altered during runtime; they include:



Save to Disk? = LabVIEW boolean input that enables data to be saved to disk.

Points to Save/Channel = LabVIEW input that determines the number of points per channel to be saved to file. The size of the resulting file is obtained multiplying this number by the number of channels. The maximum number of pionts saved to disk is only dependant on the drive's free space.

Stream File = LabVIEW file path input that specifies the name of the file to contain the recorded data.

Trigger Source = LabVIEW input used to specify the channel number used as a trigger source.

Trigger Level = LabVIEW input that determines the desired threshhold voltage level to cause a trigger.

Trigger Slope = LabVIEW input that determines the slope of the trigger.

Pretrig/Delay = LabVIEW input that takes in a negative value for pretriggering and positive values for postriggering, this value is limited by the amount allotted in 'Scope Buffer Size' input variable.

PQ_Analog Channel Array = PQ input array that specifies the PQ input sources to be displayed or recorded to disk.

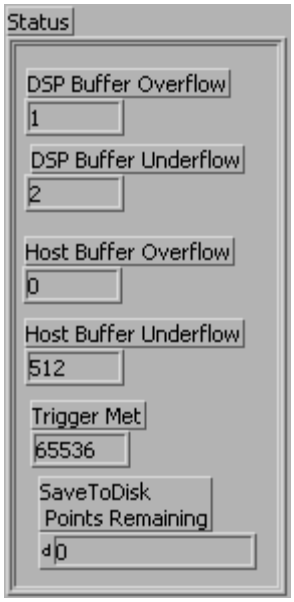
PQ_Enable = PQ input that enables the collection of data within the DSP's buffer, similar to a 'gate' input. The DSP's buffer is enabled when the queueing or PQ value is "1" or "True", and disabled when the PQ value is "0" or "False.

Note: Be aware that a queueing or PQ value is NOT a LabVIEW constant, but rather a constant within the DSP's domain which can only be sourced from a PQ variable.

Voltage Data Float = LabVIEW 2D output array representing all channel data as 32 bit, single precision, floating point values.
Note: Only applicable to the PQ_Synchronous Scope.vi.

2D I16 Data = LabVIEW 2D output array representing all channel data as 16 bit, integer values, without any scaling performed within the DSP domain. This array can be easily converted to voltage values by simply applying the appropriate scaling factors.
Note: Only applicable to the PQ_Synchronous Packed Scope.vi.

Status = LabVIEW output cluster that contains various status variables of interest; they include:



DSP Buffer Overflow = LabVIEW output that indicates when the DSP buffer was overflowed, which occurs when the DSP collects data at a higher rate than that transferred to the host, thus causing the host to drop data or become out of synchronization. In many cases, increasing the DSP buffer size will help solve this problem.

DSP Buffer Underflow = LabVIEW output that indicates when the DSP has underflowed, which occurs when the DSP collects data at a slower rate than that

requested to be transferred to the host. It occurs more frequently when selected blocks are relatively large or if the sample rate is slow relative to the transfer rate. This status indicator is benign as no data is lost.

Host Buffer Overflow = LabVIEW output that indicates that more points have been transferred to the host than it is able to display. In many cases, increasing 'Host Buffer Size' (in Setup Params) variable will help solve this problem.

Host Buffer Underflow = LabVIEW output that indicates when the amount of transferred data is insufficient relative to the requested amount of data to be displayed (selected by Display Samples/Chan). It occurs more frequently when the sample rate is relatively low, as data is shown as it is collected. This status indicator is benign as no data is lost, it is merely intended to indicate that there are insufficient points to be displayed on the screen.

Trigger Met = LabVIEW output that indicates when all trigger conditions are met.

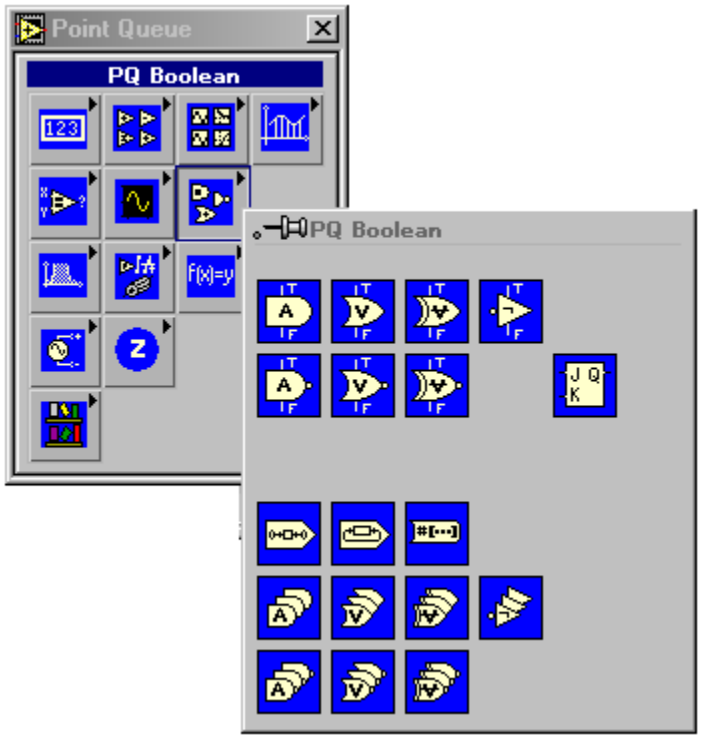
Save To Disk Points Remaining = LabVIEW output that indicates the remaining number of points per channel to be saved to disk.

4.7 PQ Boolean.

These VIs perform traditional boolean logic in two different forms:

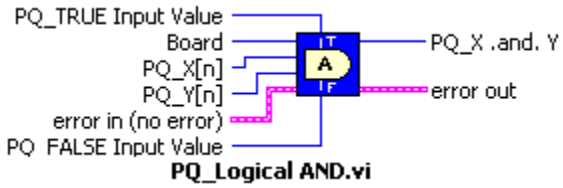
1) Logic gates where each PQ input terminal is treated as a single logical value, and whose resultant, as in the PQ comparison VIs, is the PQ TRUE/FALSE as defined by a separate pair of queueing inputs. The output returns the values wired from the PQ TRUE/FALSE terminals, where the TRUE and FALSE states may be defined with unique PQ values.

2) Logic gates where each PQ input terminal is treated as a 24 bit word with each bit representing a unique logical value, and whose resultant is also reflected as a 24 bit word. The TRUE/FALSE states are defined as traditional "0" or "1" values respectively. Because of the DSP's floating point format, single PQ terminals are limited to an effective 24 bits maximum. For operations requiring wider widths, data must be split into smaller portions with each portion not exceeding the 24 bit limit.



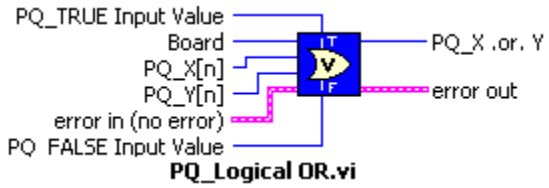
4.7.1 PQ Logical AND.

Performs a Logical AND on two PQ inputs, and return PQ True/False values.



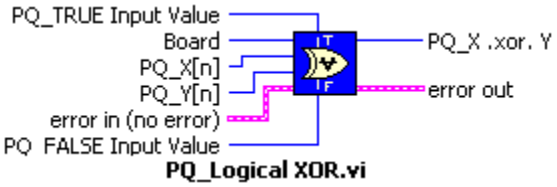
4.7.2 PQ Logical OR.

Performs a Logical OR on two PQ inputs, and return PQ True/False values.



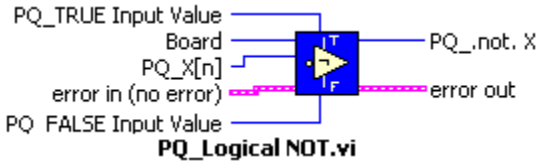
4.7.3 PQ Logical XOR.

Performs a Logical Exclusive OR on two PQ inputs, and return PQ True/False values.



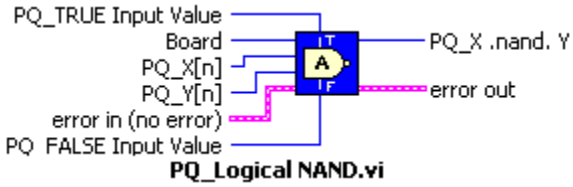
4.7.4 PQ Logical NOT.

Performs a Logical NOT or inverts the assertion level on a single PQ input.



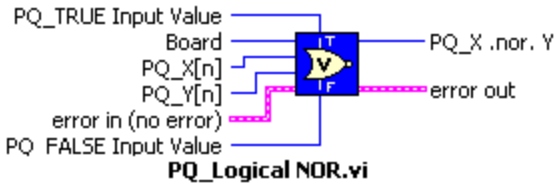
4.7.5 PQ Logical NAND.

Performs a Logical NAND on two PQ inputs, and return PQ True/False values.



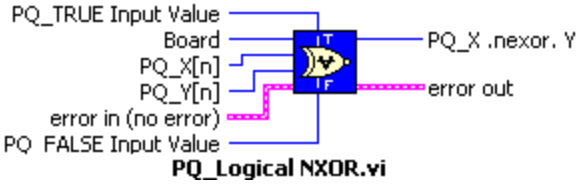
4.7.6 PQ Logical NOR.

Performs a Logical NOR on two PQ inputs, and return PQ True/False values.



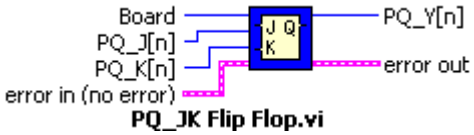
4.7.7 PQ Logical NOT XOR.

Performs a Logical NeXOR or NOT Exclusive OR on two PQ inputs, and return PQ True/False values.



4.7.8 PQ_JK Flip Flop.

This function models the behavior of a JK flip-flop. The terminals are as follows:

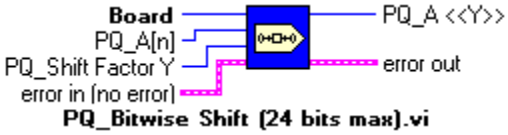


PQ_J[n] = PQ input, J terminal of JK flip-flop
PQ_K[n] = PQ input, K terminal of JK flip-flop

PQ_Y[n] = PQ output of flip-flop.

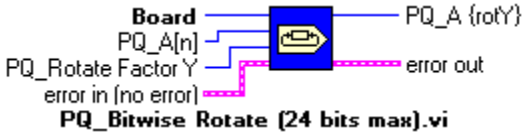
4.7.9 PQ_Bitwise Shift (24 bits max).

Performs a Bitwise Shift on the single PQ input, with the Shift factor determining both the direction and the amount of the PQ input to shift. The PQ variables represent a 24 bit integer.



4.7.10 PQ_Bitwise Rotate (24 bits max).

Performs a Bitwise Rotation on the single PQ input, with the Rotate factor determining both the direction and the amount of the PQ input to rotate. The PQ variables represent a 24 bit integer.

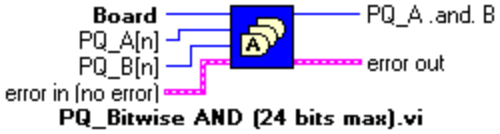


4.7.11 PQ_Bitwise Extraction (24 bits max)

Performs a Bitwise Extraction on the single PQ input, and returns a PQ output fomatted as a 1D array. Each element of the PQ output array represents the value of each bit, with the nth position of the array corresponding to the nth position of the bit it represents. The PQ variables represent a 24 bit integer.

4.7.12 PQ Bitwise AND (24 bits max).

Performs a Bitwise AND on each bit of the two corresponding PQ inputs, and returns a single PQ output with each bit representing the corresponding result. Each PQ variable represents a 24 bit integer.



4.7.13 PQ Bitwise OR (24 bits max).

Performs a Bitwise AND on each bit of the two corresponding PQ inputs, and returns a single PQ output with each bit representing the corresponding result. Each PQ variable represents a 24 bit integer.

4.7.14 PQ Bitwise XOR (24 bits max).

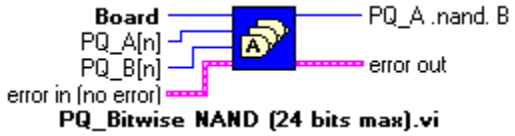
Performs a Bitwise Exclusive OR on each bit of the two corresponding PQ inputs, and returns a single PQ output with each bit representing the corresponding result. Each PQ variable represents a 24 bit integer.

4.7.15 PQ Bitwise NOT (24 bits max).

Performs a Bitwise NOT or inverts the assertion level of each bit of the single PQ input, and returns a single PQ output with each bit representing the corresponding result.

4.7.16 PQ Bitwise NAND (24 bits max).

Performs a Bitwise NAND on each bit of the two corresponding PQ inputs, and returns a single PQ output with each bit representing the corresponding result. Each PQ variable represents a 24 bit integer.



4.7.17 PQ Bitwise NOR (24 bits max).

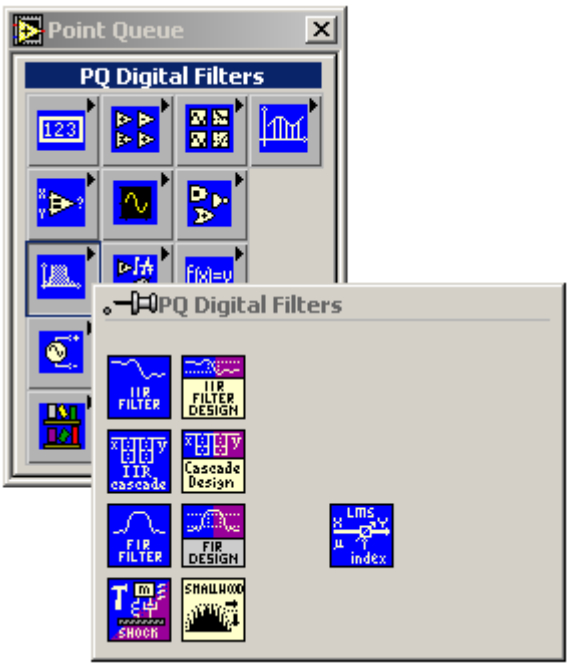
Performs a Bitwise NOR on each bit of the two corresponding PQ inputs, and returns a single PQ output with each bit representing the corresponding result. Each PQ variable represents a 24 bit integer.

4.7.18 PQ Bitwise NOT XOR (24 bits max).

Performs a Bitwise NOT Exclusive OR on each bit of the two corresponding PQ inputs, and returns a single PQ output with each bit representing the corresponding result. Each PQ variable represents a 24 bit integer.

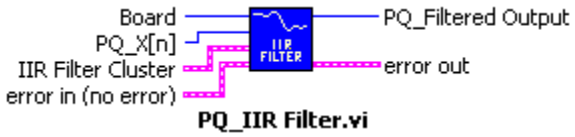
4.8 PQ Digital Filters.

These filters derive their coefficients from a separate conventional LabVIEW VI that is fed into a queuing VI. The LabVIEW VI is especially useful for finding coefficients because processing speeds are not as critical, and a large variety of filter functions already exist. It is very convenient to be able to define filter characteristics with a visual display of an impulse response in its time, frequency and phase graphs.



4.8.1 PQ_IIR Filter.

The IIR filters are setup as two separate VIs: a conventional LabVIEW based filter design VI and a DSP PQ_IIR Filter VI. The PQ_IIR Filter VI is implemented on the DSP using the well known summation of two terms or recursive formula. Please refer to the Analysis Library Reference Manual.



PQ_X[n] = PQ input to be filtered.

IIR Filter Cluster = LabVIEW input cluster containing the forward and reverse coefficients. The cluster may be derived by using one of the outputs of the LabVIEW based design VIs.

PQ_Filtered Output = PQ output that returns a continuos filtered value without the need for initial conditions, analogous to a conventional analog filter.

4.8.1.1 IIR Direct Filter Coefficients Design.

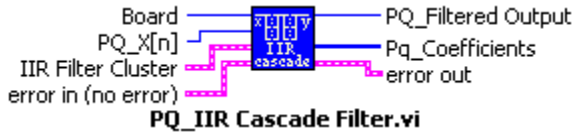
To be used as a companion VI with the PQ_IIR Filter VI.

IIR Filter Characteristics = LabVIEW input cluster to define the IIR filter characteristics.

IIR Filter Cluster = LabVIEW cluster containing the forward and reverse coefficients.

4.8.2 PQ_IIR Cascade Filter.

The IIR filters are setup as two separate VIs: a conventional LabVIEW based filter design VI and a DSP PQ_IIR Cascade Filter VI. The PQ_IIR Cascade Filter VI is implemented on the DSP using the well known summation of two terms or recursive formula. Please refer to the Analysis Library Reference Manual.



PQ_X[n] = PQ input.
IIR Filter Cluster = LabVIEW input cluster containing the forward and reverse coefficients. The cluster may be derived by using one of the outputs of the LabVIEW based design VIs.

PQ_Filtered Output = PQ output that returns a continuous filtered value without the need for initial conditions, analogous to a conventional analog filter.
PQ_Coefficients = PQ output that simply mirrors in the PQ domain the actual coeficients used by the cascade algorithm.

4.8.2.1 IIR Cascade Filter Coefficients Design.

To be used as a companion VI with the PQ_IIR Cascade Filter VI.

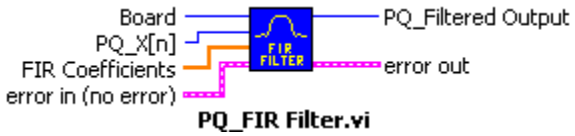
IIR Filter Characteristics = LabVIEW input cluster to define the IIR filter characteristics.

IIR Filter Cluster = LabVIEW cluster containing the forward and reverse coefficients.

4.8.3 PQ_FIR Filter.

The FIR filters are setup as two separate VIs: a conventional LabVIEW based filter design VI and a PQ_FIR Filter VI.

The PQ_FIR Filter VI is implemented on the DSP using the well known finite convolution or recursive formula. Please
refer to the Analysis Library Reference Manual.



PQ_X[n] = PQ input.
FIR Filter Coefficients = LabVIEW array containing the FIR coefficients. The array may be derived by using one of the outputs of the LabVIEW based design VIs.

PQ_Filtered Output = PQ output that returns a continuous filtered value without the need for initial conditions, analogous to a conventional analog filter.

4.8.3.1 FIR Filter Coefficients Design.

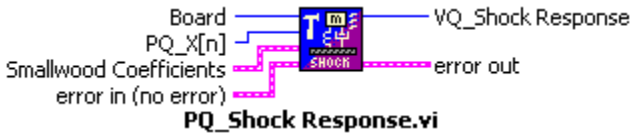
The LabVIEW based PQ_FIR Filter Design VI is merely an example of a way to design and build an array of FIR coefficients. An array of FIR coefficients can be built alternatively by using the FIR filter VIs supplied with the LabVIEW Analysis Library. However, the diagram is much cleaner when used with the supplied single icon PQ_FIR Filter Design VI.

FIR Filter Characteristics = LabVIEW input cluster to define the FIR filter characteristics.

FIR Filter Cluster = LabVIEW output cluster containing the forward coefficients.

4.8.4 PQ Shock Response.

The Shock Response takes in an array of LabVIEW coefficients, performs a Smallwood filter on many frequencies, and returns the Maxi-Max response for each frequency.



PQ_X[n] = PQ input
Smallwood Coefficients = LabVIEW bundle of two arrays. These arrays are the a and b coefficients for the Smallwood filter. The bundle of coefficients is taken from the

Smallwood coefficient generator.

VQ_Y[n] = VQ output. Each point in the outgoing vector corresponds with the array of frequencies wired into the Smallwood Coefficients terminal.

4.8.4.1 Smallwood Coefficients.

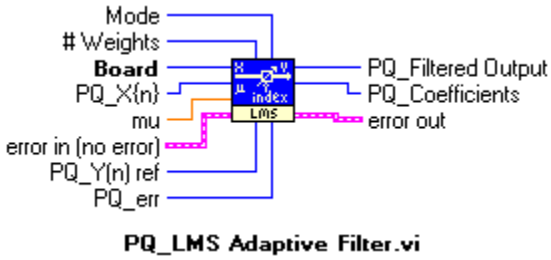
Used as a companion VI to the PQ_Shock Response VI.

Frequencies = A LabVIEW array of desired frequencies.
Q = LabVIEW input that defines the quantization level.
dT = LabVIEW input defining the sample period of the input in question.

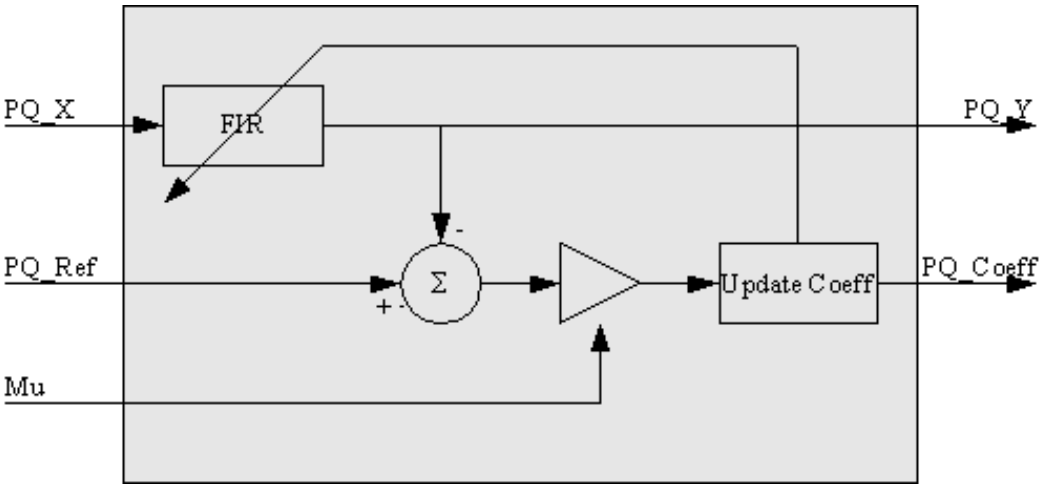
Smallwood Coefficients = A LabVIEW output cluster of the "a" and "b" coefficients for a Smallwood filter, to be wired into the Shock Response VI.

4.8.5 PQ_LMS_Adaptive Filter.

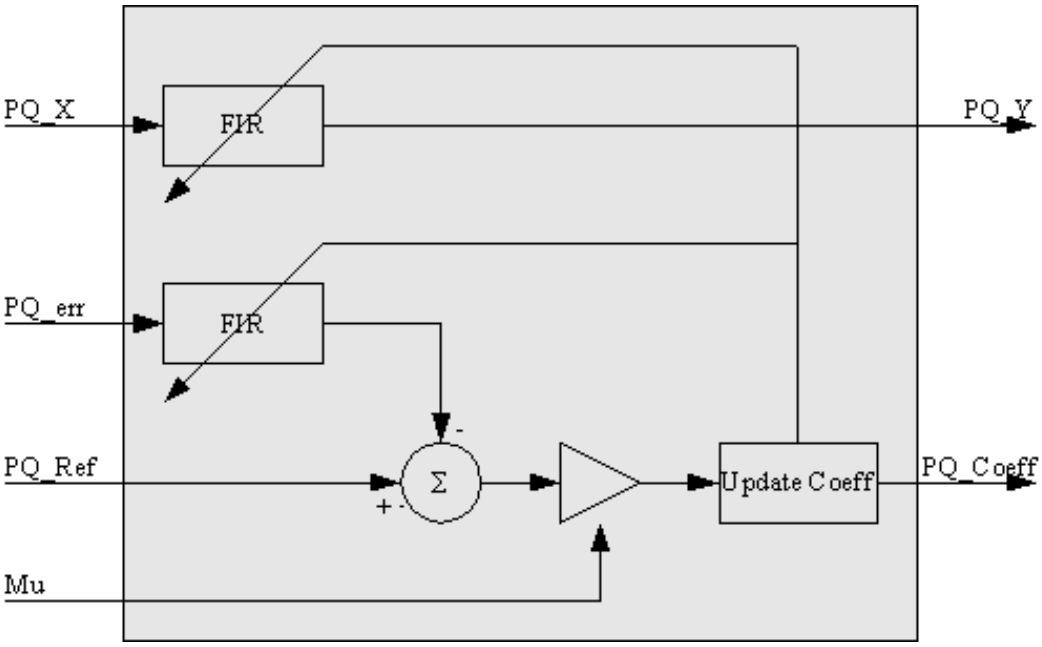
This VI performs an adaptive filtering algorithm using the Least Mean Squared method. The data that is input at PQ_X(n) is passed through the filter created by PQ_Y(n)(ref) and PQ_err. The PQ_coefficients output is an array of values that must be read using the 'DSP READ' function described above.



Mode = LabVIEW input, selects the mode of operation.
0 = Single Convolution Ignore PQ_Err: Ignores the PQ_Error input.
Below is a block diagram:



1 = Double Convolution Use PQ_Err: Uses the PQ Error input. Below is a block diagram:



 # Weights = LabVIEW input that defines the number of weighting values or coefficients to be used.

PQ_X[n] = PQ Input signal source.

mu = LabVIEW input to define the 'mu' weighting variable.

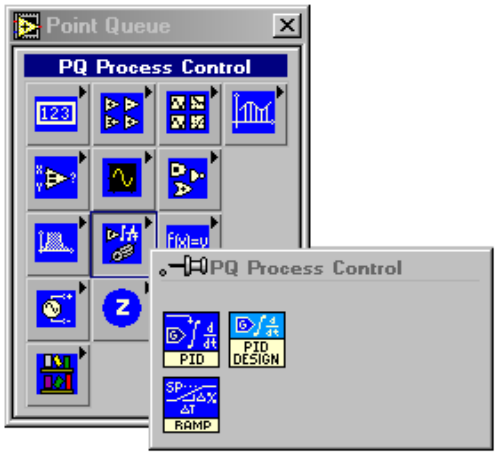
PQ_Y[n] ref = PQ input, reference source signal.

PQ_err = PQ input, error input.

PQ_Filtered Output = PQ output that returns a continuous filtered value without the need for initial conditions, analogous to a conventional analog filter.

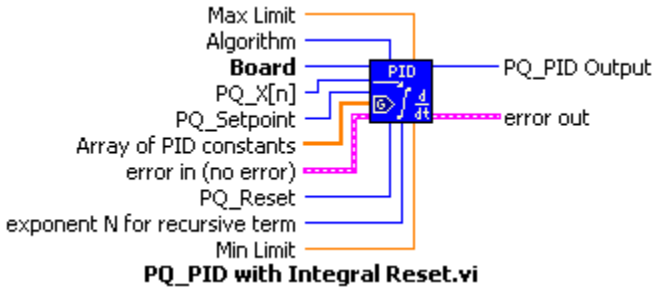
PQ_Coefficients = PQ output that simply mirrors in the PQ domain the actual coeficients used by the LMS algorithm.

4.9 PQ Process Control.



4.9.1 PQ_PID with Integral Reset.

The PQ_PID Controller VI is implemented on the DSP using the well known Proportional Integral Derivative algorithm.



PQ_X[n] = PQ input derived from the controlled system output.

PQ_Setpoint = PQ input used to pass on the desired setpoint of the controlled system in question.

Max Limit = LabVIEW input that limits the maximum output value, used to avoid output saturation if system becomes unstable.

Min Limit = LabVIEW input that limits the minimum output value, used to avoid output saturation if system becomes unstable.

Array of PID constants = LabVIEW array of constants that are computed from the supplied PID Design VI whose inputs are Kp, Ki, and Kd.

Algorithm = LabVIEW input that determines if the algorithm is direct or derived from the Z transform.

Direct: If set for the direct algorithm, each term is computed discretely, rendering the same effect as if one were to use discrete function icons to represent each term (Subtraction, Integral, Derivative). This method is recommended for most

applications.

Z Transform: If set for the Z transform algorithm, the function uses the equation derived by first taking the classical continuous time domain expression, then transforming it to the continuous frequency domain via the Laplace transform, then to the discrete frequency domain via the Z transform, and finally to the discrete time domain, or f(t) -> f(s) =>> F(z) -> F(n) yields:

u[n] = u[n-N] + K1{e[n]} + K2 {e[n-1]} + K3{e[n-2]}

where "N", either a '1' or a '2', represents the exponent of the non weighted 'z' term, which in turn is manifest as the amount of delay in the non weighted recursive term of the discrete time domain equation. This exponent will range from '1' or '2' depending upon the 'z' approximation used when substituting for the 's' variable.

Finally, note that the Kp, Ki, and Kd constants are reevaluated as determined by the sample period to yield the new K[3:1] constants.

***NOTE**:*
1) The 'Z transform' algorithm is recommended for advanced users only, where the companion 'PID Design' VI must be used.
2) Please note that there are several approximations for the 'z' domain variable when substituting for Laplace's 's' variable. Please consult the 'PID Design' VI for more details.

Exponent N for recursive term = LabVIEW constant that determines the amount of delay of the non weighted recursive term. The value of '1' or '2' represents the exponent of the non weighted 'z' term, which in turn is manifest as the amount of delay in the non weighted recursive term of the discrete time domain equation. This exponent will range from '1' or '2' depending upon the 'z' approximation used when substituting for the 's' variable.

***NOTE:** Not applicable for the 'Direct' algorithm.*

PQ_PID Output = PQ command output to be sent to the controlled system input.

4.9.1.1 PID Design.

Used as a companion VI to the 'PQ_PID with Integral Reset' VI..

PID Parameters = LabVIEW input cluster to define the PID characterisitics.

Sample Rate: LabVIEW input, required to compute the sample period.

Kp: LabVIEW input, constant that determines the weight of the proportional term.

Ki: LabVIEW input, constant that determines the weight of the integral term.

Kd: LabVIEW input, constant that determines the weight of the derivative term.

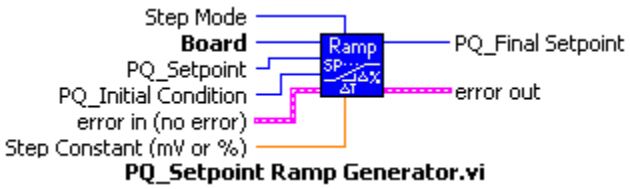
Approximation Method: LabVIEW input, selects the 'z' domain approximation when substituting for Laplace's 's' variable. The choices are:

1) None: best used for the direct algorithm
2) Backward Difference: One of three choices to compute the PID constants when using the 'Z Transform' algorithm.
3) Trapezoidal/Tustin/Bilinear: One of three choices to compute the PID constants when using the 'Z Transform' algorithm.
4) Rectangular: One of three choices to compute the PID constants when using the 'Z Transform' algorithm.

Array of PID constants = LabVIEW output array containing the PID coefficients to be used by the DSP.

4.9.2 PQ_Setpoint Ramp Generator.

The PQ_Setpoint Ramp Generator function converts a step signal to a ramp whose gradient is determined by the Step Constant. When a new setpoint is entered, an undesired transient may take place during the transition with the new setpoint value. The smaller the Step Constant value, the smaller the gradient will be in approaching the final setpoint value.



PQ_Setpoint = PQ input used to pass on the desired setpoint of the controlled system in question.

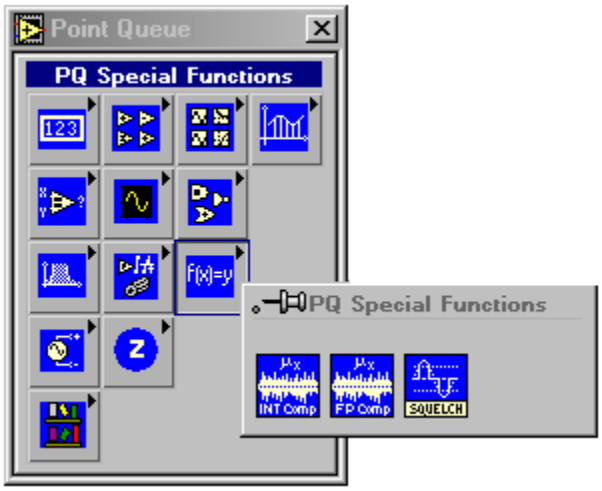
PQ_Initial Condition = PQ input used to set the initial value from which to start the ramp.

Step Mode = LabVIEW input that determines if the Step Constant is to be expressed as units of mV or a percentage between the initial and final points.

Step Constant = LabVIEW input used to determine the gradient of the slope of the PQ_Final Setpoint output. If the Step Mode is set for mV, each sample will increment or decrement by this mV value until the final setpoint is reached. If the Step Mode is set for percentage, each sample will be incremented to a proportion equal to the percent of the difference between the current or initial value and the final setpoint value.

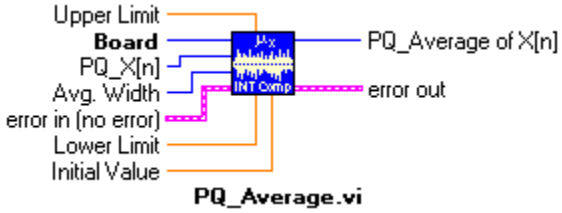
PQ_Final Setpoint = PQ output used to pass on the ramped setpoint.

4.10 PQ_Special Functions.



4.10.1 PQ_Average.

The PQ_Average function computes and returns a moving average of the input data points based on the Width variable. Therefore, the moving average output will behave similar to a lowpass filter.



PQ_X[n] = PQ input.

Avg. Width = LabVIEW constant to determine the number of points from which to compute the moving average.

Upper Limit = LabVIEW input, defines the upper limit/clipping value if the 'wind-up' phenomenon is detected.

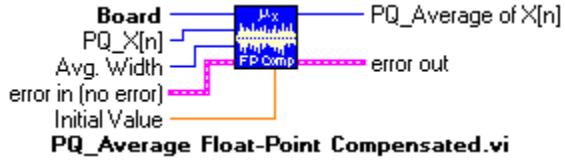
Lower Limit = LabVIEW input, defines the lower limit/clipping value if the 'wind-up' phenomenon is detected.

Initial Value = LabVIEW input, initial condition.

PQ_Average of X[n] = PQ output reflecting the moving average on a block of "N" points.

4.10.2 PQ_Average Float-Point Compensated.

The 'PQ_Average Float-Point Compensated.vi' function computes and returns a moving average of the input data points based on the Width variable. The mantissa portion of the floating point result is compensated so as to nullify the 'wind-up' phenomenon, caused by the limited floating point resolution of the DSP. The moving average output will behave similar to a lowpass filter.

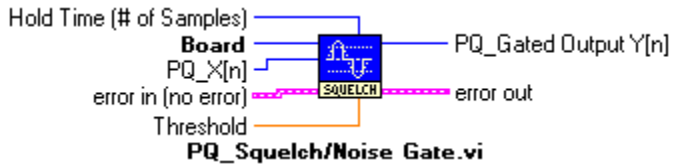


PQ_X[n] = PQ input.
Avg. Width = LabVIEW constant to determine the number of points from which to compute the moving average.
Initial Value = LabVIEW input, initial condition.

PQ_Average of X[n] = PQ output reflecting the moving average on a block of "N" points.

4.10.3 PQ_Squelch/Noise Gate.

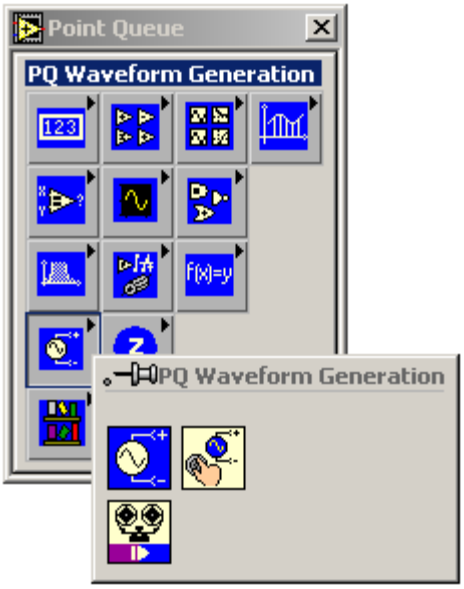
This VI behaves much like a CB radio squelch control. It reduces the output level to 'zero' when the input signal falls below a predefined threshold.



PQ_X(n) = PQ input.
Hold Time = LabVIEW input representing the number of samples to hold the gate open after the input signal falls below the threshold.
Threshold = LabVIEW input, minimum signal level to open the gate

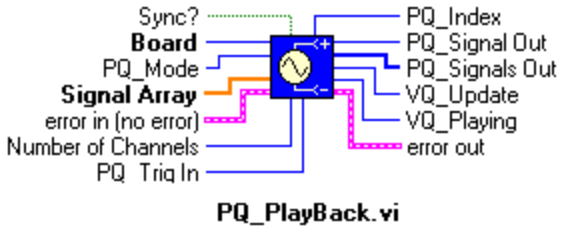
PQ_Gated Output = PQ output.

4.11 Waveform Generation.



4.11.1 PQ_PlayBack.

The PQ_Playback VI allows the DSP to constantly play back an array of values, useful for the creation of function generators.



Signal Array = LabVIEW array input used to generate a signal derived from a conventional LabVIEW VI, such as the PQ_Waveform Design VI.
***NOTE**: As with filter design VIs, the LabVIEW based 'PQ_Waveform Design' VI is merely an example of a way to design and build an array containing a conventional signal. An array for any signal can be built alternatively by using the function pattern VIs supplied with the LabVIEW Analysis Library. However, the diagram is much cleaner when used with the supplied single icon 'PQ_Waveform Design' VI.*

PQ_Mode = PQ input which selects the output waveform to be continuous or single shot.

Sync? = When this boolean is set to F, the Signal Array will be updated Asynchronously. When TRUE, the VI will wait until the current array has finished playing before updating the waveform. If the output waveform is static, i.e. a sine table, then this should be set to FALSE, which will provide better performance. If the waveform is arbitrary, and constantly changing, i.e. a voice recording, then Sync should be set to TRUE. This can slow down the VI since it must wait until the previous array has finished playing.

PQ_Trig In = PQ input when set to a PQ zero, enables the output to be updated. Any PQ nonzero values disable updating of outputs. Default to PQ zero if not wired.

Number of Channels = Allows for an array of duplicate output waveforms to be made available on the PQ_Signal Out array terminal. Useful in the case where multiple outputs need to be simultaneously reproduced.

PQ_Index = PQ output that returns the current pointer position of the waveform buffer.

PQ_Signal Out = PQ output used to wire to a source needing a waveform, such as the PQ_Analog Output VI.

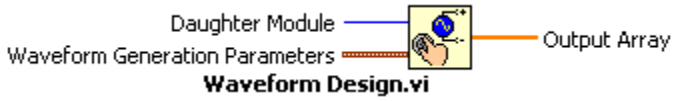
PQ_Signals Out = PQ output, similar to PQ_Signal Out, but has has multiple interleaved signals. Useful in designed needing various function generator outputs.

VQ_Update = VQ output used by the PQ_Waveform Playback VI to synchronize the buffer pointer currently being updated, useful for playback of data from a file.

VQ_Playing = VQ output used by the PQ_Waveform Playback VI to synchronize the buffer pointer currently being played back, useful for playback of data from a file.

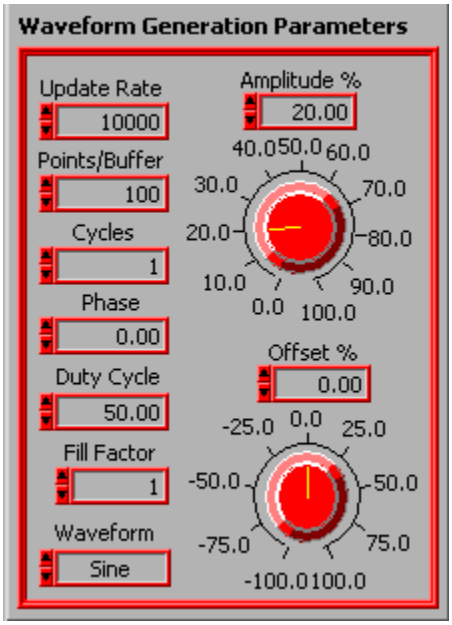
4.11.2 Waveform Design.

To be used as a conmpanion to the PQ_Playback VI.



Output Array = LabVIEW output array containing all the values to build the desired waveform.

Waveform Generation Parameters = A LabVIEW input cluster of Controls Parameters to generate the desired Waveform Array, typical control is shown below:



Parameters included to define a waveform are as follows:

Update Rate = DAC update rate, only used for computational purposes.

Points / Buffer = Number of points to include in the resulting buffer (array) containing the desired waveform.

Cycles (Default 1) = Nuber of cycles per buffer (repetitions) of a waveform.

Phase (Default 0) = Phase in radians that defines the waveform starting point. Defaulted ata zero vlue.

Duty Cycle (Default 50) = Ratio of the positive portion of a waveform; only applicable to the SQUARE wave.

Fill Factor (Default 1) = The fill factor inserts "dumby" points or multiples of the same points that define a waveform. Usedful when an output is fed back to an input, where it is desired to measure th response of signal of several multiple of output samples. The fill factor effectively increases the buffer size by multiplying it times the buffer size.

Waveform (Default Sine) = Waveform selection, user can select a standard waveform pattern: Sine, Square, Triangle, Sawtooth, Exponent, Sync, Pulse, Noise.

Amplitud % = Amplitude of desired signal.

Offset = Constant biasing value to be added to resulting signal.

4.11.3 PQ Waveform Playback. (Advanced Users)

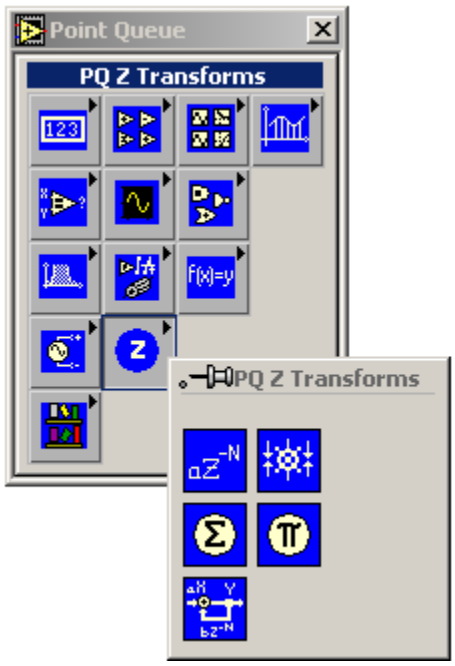
This VI is used when creating a non-requeueing VI that employs a PQ_PlayBack function. This VI would normally reside in the LabVIEW While Loop in order to continually update the output waveform. Particularly useful for playback of data from a file in real time without gaps.

VQ_Update = VQ input derived from the PQ_Playback VI, used to synchronize the buffer pointer currently being updated.

VQ_Playing = VQ input derived from the PQ_Playback VI, used to synchronize the buffer pointer currently being played.

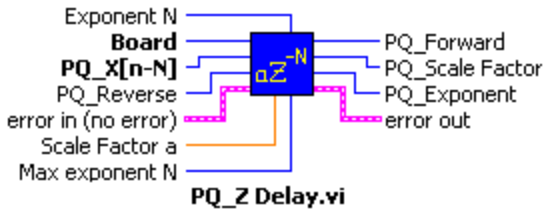
Signal Array = LabVIEW array input used to generate a signal derived from a binary data source such as a conventional LabVIEW VI, or the PQ_Waveform Design VI.

4.12 Z Transforms.



4.12.1 PQ_Z Delay.

The PQ_Z Delay will delay a queuing input by the LabVIEW constant defined by the "N" delay variable. The PQ_Feedback Node VI is used in conjunction with the PQ_Z Delay in order to satisfy LabVIEW's requirement of data dependency. The PQ_Z Delay and PQ_Feedback Node subVIs can be used to form feedback loops when creating a lattice structure. This flexibility is especially useful for performing unconventional transfer functions.



PQ_X[n-N] = PQ input to be delayed by "exponent N" sample periods, which is exclusively derived from the PQ_Feedback Node VI output of the same name. Because LabVIEW does not allow for recursive wiring, this delayed queuing input variable cannot be fed backward and directly into another queuing input.

PQ_Reverse = PQ input to be delayed by "exponent N" sample periods, and used only in the case that the delayed sample is fed forward. In this case, data dependency is maintained and no intermediate VIs are necessary.

Exponent N = LabVIEW input to determine the number of sample periods to delay a PQ input variable.

Scale Factor a = LabVIEW input that directly multiplies the PQ input variable in question.

Max exponent N = LabVIEW input that defines maximum number of delays.

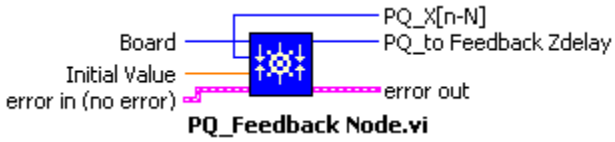
PQ_Forward = PQ output, reflecting the delayed sample input.

PQ_Scale Factor = LabVIEW constant that is multiplied by the "Z" value.

PQ_Exponent = PQ returned exponent. Theoretically it must be same as Exponent N, but because we have a limit (Max exponent N), this output is used as an indicator to the subsequent VIs following as to the N value actually used.

4.12.2 PQ_Feedback Node.

Most Z Transform implementations require feedback to represent recursive terms. LabVIEW's graphical compiler does not support direct feedback. In order to overcome this problem, the PQ_Feedback Node VI was created as a companion VI to the PQ_Z Delay VI in backward mode. Therefore, when using the PQ_Z Delay VI in backward mode, a feedback loop can be implemented in conjunction with the companion PQ_Feedback Node VI. To do this, the output terminal of the PQ_Z Delay (PQ_X[n-N]) is wired to the same terminal of the PQ_Feedback Node VI. The PQ_X[n-N] terminal is then wired to any queuing input of a QuVIEW function; typically, the input of the PQ_Summing Node VI.



Initial Value = LabVIEW input that is added to the delayed queuing variable in question.

PQ_X[n-N] = PQ input to be delayed by "exponent N" sample periods, which is

exclusively fed into the PQ_Z Delay VIs queuing input of the same name. Because LabVIEW does not allow for recursive wiring, this delayed queuing input variable cannot be fed backward and directly into another queuing input.

PQ_to Feedback Zdelay = PQ output that must be connected to the 'PQ_Reverse' terminal of the 'PQ_Z Delay VI' when normal feedback is needed. This is used as a method to circumvent LabVIEW's restriction on recursive feedback wires.

4.12.3 PQ_Summing Node.

PQ_Array of X[i] = PQ input array that is added and returned as a single value. The array format is derived by using a conventional LabVIEW build array icon, whose inputs are wired from one or more PQ output variables.

PQ_Sum of Array = PQ output whose value is the sum of the array of PQ inputs.

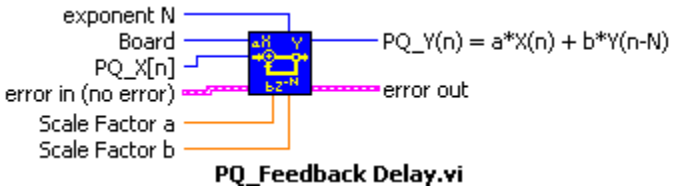
4.12.4 PQ_Multiplying Node.

PQ_Array of X[i] = PQ input array whose elements are multiplied and returned as a single value, similar in concept to a factorial. The array format is derived by using a conventional LabVIEW build array icon, whose inputs are wired from one or more PQ output variables.

PQ_Product of Array = PQ output whose value is the product of the array of PQ inputs.

4.12.5 PQ_Feedback Delay.

The PQ_Feedback Delay VI will delay a queuing output by the LabVIEW constant defined by the "N" delay variable. The reason for having an extra VI perform this function is because LabVIEW forces data dependency on the program structure. Therefore, a delayed output cannot be added onto itself unless this unique VI is used.



PQ_X[n] = PQ input to be delayed by "N" sample periods.

Exponent N = LabVIEW input to determine the number of sample periods to delay a PQ input variable. Scale

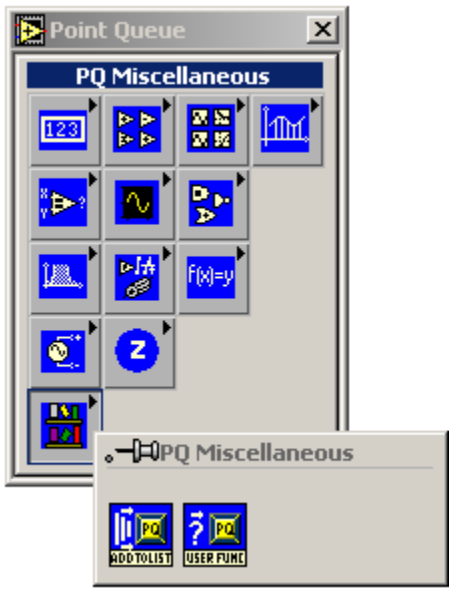
Factor a = LabVIEW input that directly multiplies the PQ input variable in question.

Factor b = LabVIEW input that is added to the result of the product of 'a * PQ_X[n]'.

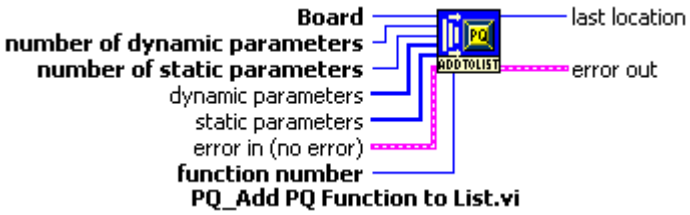
PQ_Y[n] = a*X[n] + b*Y[n-N] = PQ output.

4.13 PQ Miscellaneous

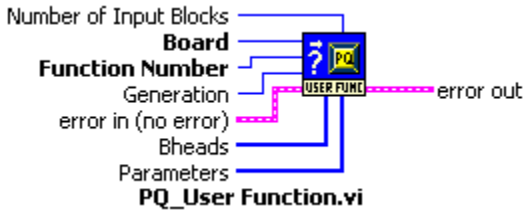
These VIs are not intended for users, and are nothing more than building block functions for QuVIEW.



4.13.1 PQ_Add PQ Function to List



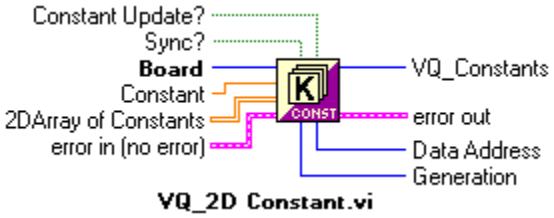
4.13.2 PQ_User Function.


Number of Input Blocks
Board
Function Number
Generation
error in (no error)
Bheads
Parameters
PQ_User Function.vi
error out

**$$$

5.1.1 VQ 2D Constants.

Transfers a 2D LabVIEW array of constants to the DSP.



Constant Update? = LabVIEW boolean input, indicates if data will be continuously downloaded to the DSP for each execution iteration or not.

Sync? = LabVIEW boolean input, indicates if execution is to be carried out Asynchronously, regardless if data is updated from the host.

Constant = LabVIEW input, a single constant value. The value wired to this terminal is used by default if the '2D Array of Constants' is left unwired

2D Array of Constants = LabVIEW input, a 2D array of constants. The array wired to this terminal overrides the single value wired into the 'Constant' terminal.

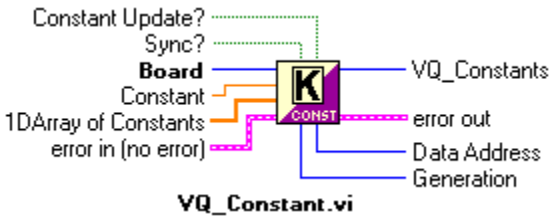
VQ_Constants = VQ output containing the constants wired in from LabVIEW.

Data Address = LabVIEW output, a DSP pointer indicating where the input data was allocated in the DSP. For advanced users only.

Generation = LabVIEW output, a generation number of the data block allocated, used by other VQ's for synchronization. For advanced users only.

6.1.2 VQ Constant.

Transfers a 1D LabVIEW array of constants to the DSP.



Constant Update? = LabVIEW boolean input, indicates if data will be continuously downloaded to the DSP for each execution iteration or not.

Sync? = LabVIEW boolean input, indicates if execution is to be carried out Asynchronously, regardless if data is updated from the host.

Constant = LabVIEW input, a single constant value. The value wired to this terminal is used by default if the '1D Array of Constants' is left unwired

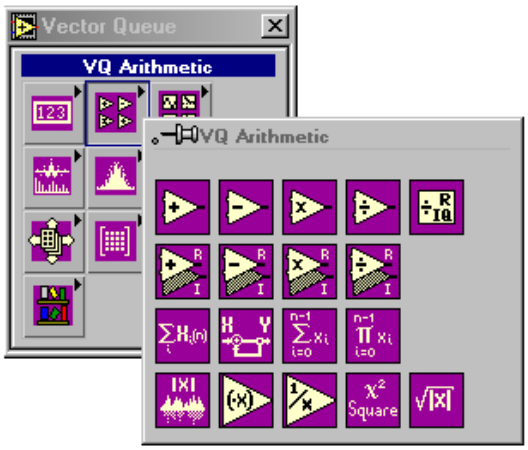
1D Array of Constants = LabVIEW input, a 1D array of constants. The array wired to this terminal overrides the single value wired into the 'Constant' terminal.

VQ_Constants = VQ output containing the constants wired in from LabVIEW.

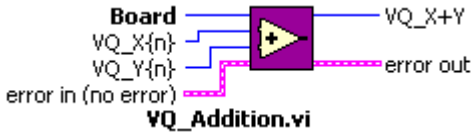
Data Address = LabVIEW output, a DSP pointer indicating where the input data was allocated in the DSP. For advanced users only.

Generation = LabVIEW output, a generation number of the data block allocated, used by other VQ's for synchronization. For advanced users only.

5.2 VQ Arithmetic Functions.



5.2.1 VQ Addition.

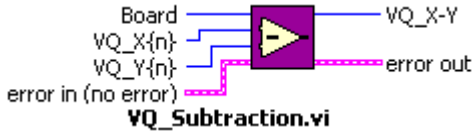


VQ_X{n} = VQ input.

VQ_Y{n} = VQ input.

PQ_X+Y = VQ output, sum of X{n} and Y{n}.

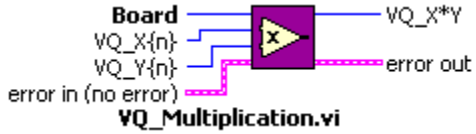
5.2.2 VQ Subtraction.



VQ_X{n} = VQ input.
VQ_Y{n} = VQ input.

VQ_X-Y = VQ output, difference of X{n} and Y{n}.

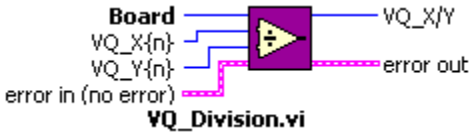
5.2.3 VQ Multiplication.



VQ_X{n} = VQ input.
VQ_Y{n} = VQ input.

VQ_X*Y = VQ output, product of X{n} and Y{n}.

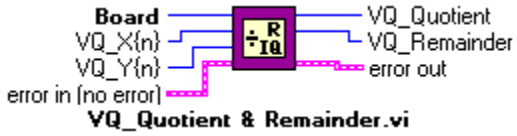
5.2.4 VQ Division.



VQ_X{n} = VQ input.
VQ_Y{n} = VQ input.

VQ_X/Y = VQ output, ratio of X{n} over Y{n}.

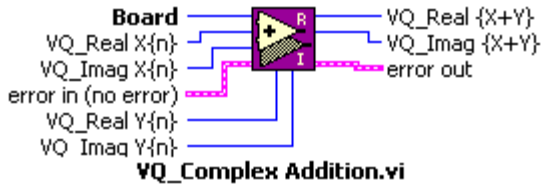
5.2.5 VQ Quotient & Remainder.



VQ_X{n} = VQ input.
VQ_Y{n} = VQ input.

VQ_Quotient = VQ output, quotient from the ratio of X{n} over Y{n}.
VQ_Remainder = VQ output, remainder from the ratio of X{n} over Y{n}.

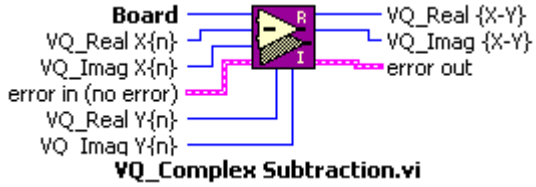
5.2.6 VQ Complex Addition.



VQ_Real X{n} = VQ input.
VQ_Imag X{n} = VQ input.
VQ_Real Y{n} = VQ input.
VQ_Imag Y{n} = VQ input.

VQ_Real X+Y = VQ output, sum of real parts of X{n} and Y{n}.
VQ_Imag X+Y = VQ output, sum of imaginary parts of X{n} and Y{n}.

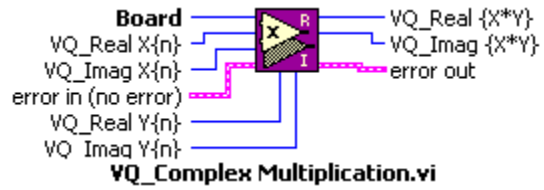
5.2.7 VQ Complex Subtraction.



VQ_Real X{n} = VQ input.
VQ_Imag X{n} = VQ input.
VQ_Real Y{n} = VQ input.
VQ_Imag Y{n} = VQ input.

VQ_Real X-Y = VQ output, difference of real parts of X{n} and Y{[n}.
VQ_Imag X-Y = VQ output, difference of imaginary parts of X{n} and Y{n}.

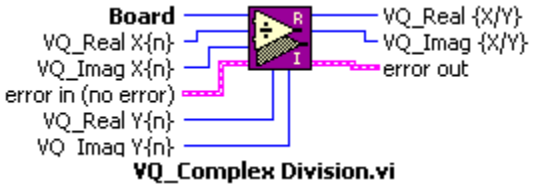
5.2.8 VQ Complex Multiplication.



VQ_Real X{n} = VQ input.
VQ_Imag X{n} = VQ input.
VQ_Real Y{n} = VQ input.
VQ_Imag Y{n} = VQ input.

VQ_Real X*Y = VQ output, real part of complex multiplication of X{n} and Y{n}.
VQ_Imag X*Y = VQ output, imaginary part of complex multiplication of X{n} and Y{n}.

5.2.9 VQ_Complex Division.



VQ_Real X{n} = VQ input.
VQ_Imag X{n} = VQ input.
VQ_Real Y{n} = VQ input.
VQ_Imag Y{n} = VQ input.

VQ_Real X/Y = VQ output, real part of complex division of X{n} and Y{n}.
VQ_Imag X/Y = VQ output, imaginary part of complex division of X{n} and Y{n}.

5.2.10 VQ_Vector Summation.

This VI returns a vector, VQ_Sum Xi{n}, equal in size to each of the vectors in the input array, VQ_Array of X{n}, which are also of the same size. Each i'th element of the result is the sum of the respective i'th elements of the input vectors; i.e., the 3rd element of VQ_Sum Xi{n} is the sum of the 3rd elements of all of the input VQ_X{n} vectors.

VQ_Array of X{n} = VQ input array.

VQ_Sum Xi{n} = VQ output whose values are the sum of each corresponding input vector element.

5.2.11 VQ_Recursive Vector Summation.

This VI returns a vector, VQ_Yi{n}, equal in size to the input vector , VQ_Xi{n}. Each

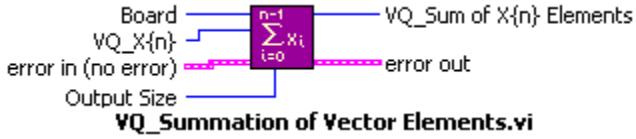
i'th element of the result is the sum of the respective i'th elements of the input vector and the previous result; i.e., the 3rd element of VQ_Yi{n} is the sum of the 3rd element of the input vector VQ_Xi{n} and the 3rd element of the previous output vector VQ_Y(i-1){n}. This algorithm is especially useful when trying to find the result of recursive averages over a set of data blocks.

VQ_Xi{n} = VQ input.

VQ_Yi{n} = Xi{n} + Yi-1{n} = VQ output whose value is the compounding sum of the current input vector elements and the sum of the previous input vector elements.

5.2.12 VQ Summation of Vector Elements.

This VI returns a vector, VQ_Sum of X{n} Elements, equal in size to the LabVIEW input, Output Size. Each i'th element of the result is the sum of all of the elements of a corresponding input vector; i.e., the 3rd element of VQ_Sum of X{n} Elements is the sum of all of the elements of the 3rd input vector VQ_X{n}. This algorithm is especially useful when trying to find a trend or behavior over a long period of time or large set of data blocks.

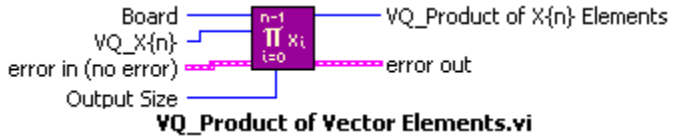


VQ_X{n} = VQ input.
Output Size = LabVIEW input. Specifies the output vector size such that each element of the VQ output is the resultant sum of each input vector element.

VQ_Sum of X{n} Elements = VQ output whose individual values are the sum of each input vector's elements. The overall length of the output vector array, specified by the LabVIEW input Output Size, represents the overall number of vectors used for the computation.

5.2.13 VQ Product of Vector Elements.

This VI returns a vector, VQ_Product of X{n} Elements, equal in size to the LabVIEW input, Output Size. Each i'th element of the result is the product of all of the elements of a corresponding input vector; i.e., the 3rd element of VQ_Product of X{n} Elements is the product of all of the elements of the 3rd input vector VQ_X{n}. This algorithm is especially useful when trying to find a trend or behavior over a long period of time or large set of data blocks.



VQ_X{n} = VQ input.

Output Size = LabVIEW input. Specifies the output vector size such that each element of the VQ output is the resultant product of each input vector element.

VQ_Product of X{n} Elements = VQ output whose individual values are the product of each input vector's elements. The overall length of the output vector array, specified by the LabVIEW input Output Size, represents the overall number of vectors used for the computation.

5.2.14 VQ_Absolute Value.

VQ_X{n} = VQ input.

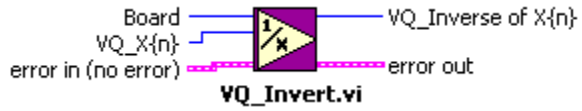
VQ_|X{n}| = VQ output, absolute value of X{n}.

5.2.15 VQ_Negate.

VQ_X{n} = VQ input.

VQ_-(X{n}) = VQ output, negative of X{n}.

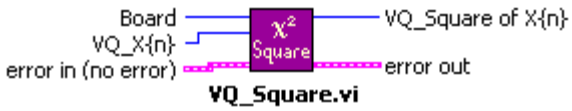
5.2.16 VQ_Invert.



VQ_X{n} = VQ input.

VQ_Inverse of X{n} = VQ output, inverse of X{n}.

5.2.17 VQ_Square.



VQ_X{n} = VQ input.

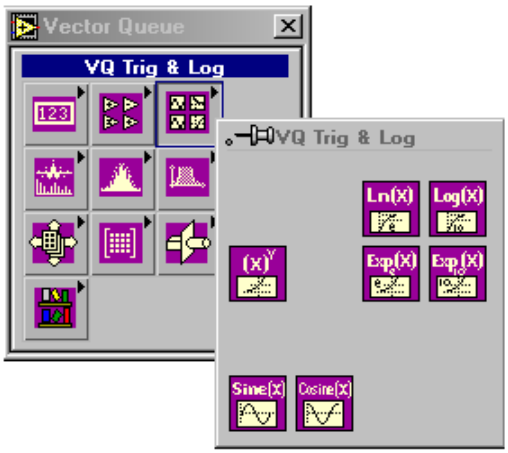
VQ_Square of X{n} = VQ output, square of input vector X{n}.

5.2.18 VQ Square Root.

VQ_X{n} = VQ input.

VQ_Square Root of |X{n}| = VQ output, square root of absolute value of X{n}.

5.3 VQ Trig and Log.



5.3.1 VQ Natural Logarithm.

VQ_X{n} = VQ input.

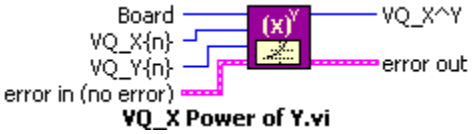
VQ_Natural Log of |X{n}| = VQ output, computes natural logarithm base "e" with argument as the absolute value of X{n}.

5.3.2 VQ Log Base 10.

VQ_X{n} = VQ input.

VQ_Log|X{n}| = VQ output, computes logarithm base "10" with argument as the absolute value of X{n}.

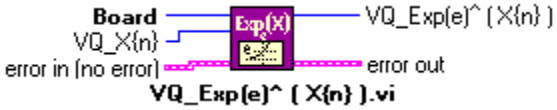
5.3.3 VQ X to Power of Y.



VQ_X{n} = VQ input.
VQ_Y{n} = VQ input.

VQ_X^Y = VQ output, computes exponential base X{n} with power of Y{n}.

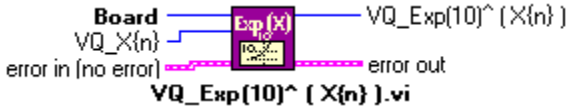
5.3.4 VQ Exp(e)^(X{n}).



VQ_X{n} = VQ input.

VQ_Exp(e)^(X{n}) = VQ output, computes exponential base "e" with power of X{n}.

5.3.5 VQ Exp(10)^(X{n}).



VQ_X{n} = VQ input.

VQ_Exp(10)^(X{n}) = VQ output, computes exponential base "10" with power of X{n}.

5.3.6 VQ_Sine.

VQ_X{n} = VQ input.

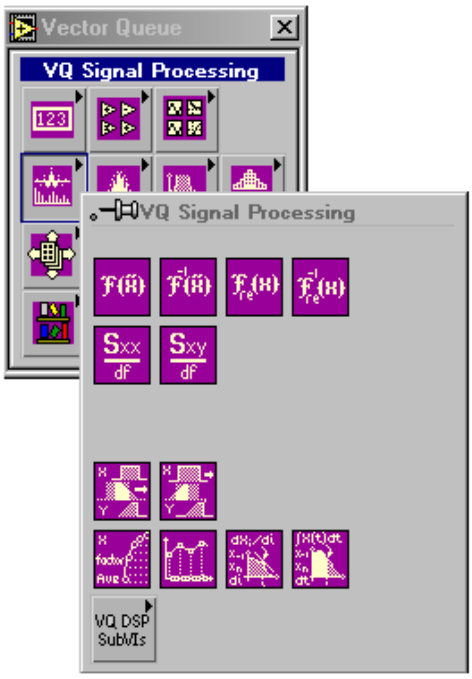
VQ_Sine(X{n}) = VQ output, computes the sine of radians expressed in the X{n} vector.

5.3.7 VQ_Cosine.

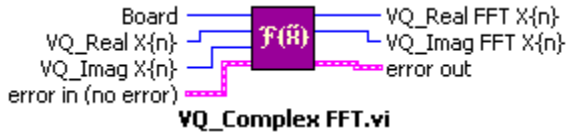
VQ_X{n} = VQ input.

VQ_Cosine(X{n}) = VQ output, computes the cosine of radians expressed in the X{n} vector.

5.4 VQ Signal Processing.



5.4.1 VQ Complex FFT.



VQ_Real X{n} = VQ input.
VQ_Imag X{n} = VQ input.

VQ_Real FFT X{n} = VQ output.
VQ_Imag FFT X{n} = VQ output.

5.4.2 VQ Inverse Complex FFT.

VQ_Real FFT X{n} = VQ input.
VQ_Imag FFT X{n} = VQ input.

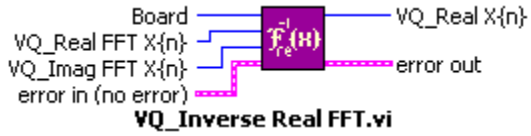
VQ_Real X{n} = VQ output.
VQ_Imag X{n} = VQ output.

5.4.3 VQ Real FFT.

VQ_Real X{n} = VQ input.

VQ_Real FFT X{n} = VQ output.
VQ_Imag FFT X{n} = VQ output.

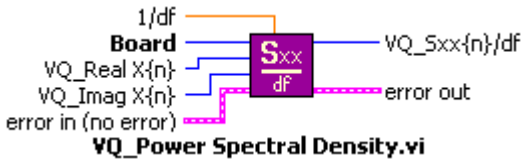
5.4.4 VQ Inverse Real FFT.



VQ_Real FFT X{n} = VQ output input.
VQ_Imag FFT X{n} = VQ output input.

VQ_Real X{n} = VQ output.

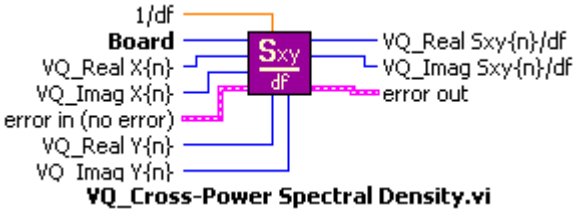
5.4.5 VQ_Power Spectral Density.



VQ_Real FFT X{n} = VQ input.
VQ_Imag FFT X{n} = VQ input.
1/df = LabVIEW input. If left unwired this defaults to 1 and returns a Power Spectrum. Otherwise, this value can be computed by Sample Rate/n.

VQ_Sxx{n}/df = VQ output that reflects the Power Spectral Density weighted against frequency.

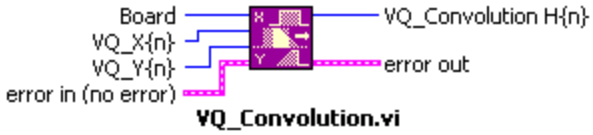
5.4.6 VQ_Cross-Power Spectral Density.



VQ_Real FFT X{n} = VQ input of the real portion of the first vector.
VQ_Imag FFT X{n} = VQ input of the imaginary portion of the first vector.
VQ_Real FFT Y{n} = VQ input of the real portion of the second vector.
VQ_Imag FFT Y{n} = VQ input of the imaginary portion of the second vector.
1/df = LabVIEW input. If left unwired this defaults to 1 and returns a Cross-Power Spectrum. Otherwise, this value can be computed by Sample Rate/n.

VQ_Real Sxy{n}/df = VQ output of the real portion of the Cross-Spectral Density.
VQ_Imag Sxy{n}/df = VQ output of the imaginary portion of the Cross-Spectral Density.

5.4.7 VQ_Convolution.



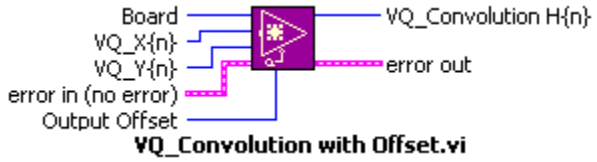
VQ_X{n} = VQ input.

VQ_Y{n} = VQ input.

VQ_Convolution H{n} = VQ output.

5.4.7.1 VQ_Convolution with Offset.

This sub-vi is only used for special cases of convolutuions, and is not intended for general use.



VQ_X{n} = VQ input.
VQ_Y{n} = VQ input.
Output Offset = LabVIEW input used to change the index of the first point after the convolution is performed. Useful for recursive functions such as IIR filters.

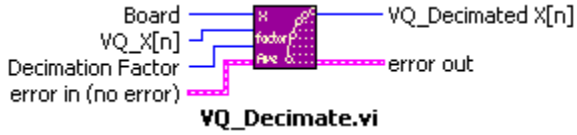
VQ_Convolution H{n} = VQ output.

5.4.8 VQ_Correlation.

VQ_X{n} = VQ input.
VQ_Y{n} = VQ input.

VQ_Convolution H{n} = VQ output.

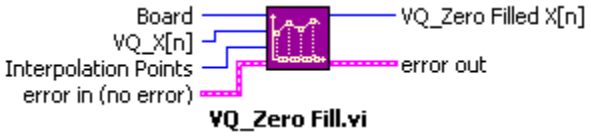
5.4.9 VQ_Decimate.



VQ_X{n} = VQ input.
Decimation Factor = LabVIEW input that determines every nth point to keep in the newly sized vector.

VQ_Decimated X{n} = VQ output.

5.4.10 VQ_Zero Fill.

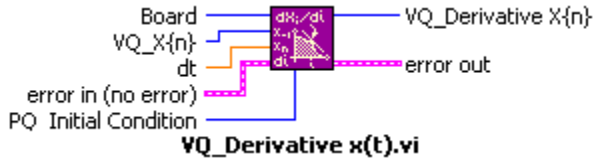


VQ_X{n} = VQ input.

Interpolation Points = LabVIEW input that determines the number of zeroes to insert between every sample.

VQ_Zero Filled X{n} = VQ output.

5.4.11 VQ Derivative x(t).



VQ_X{n} = VQ input.

dt = LabVIEW input that determines width at which to divide the current and previous values of X{n}.

VQ_Initial Condition = VQ input, initial condition used to calculate first value.

VQ_Derivative X{n} = VQ output.

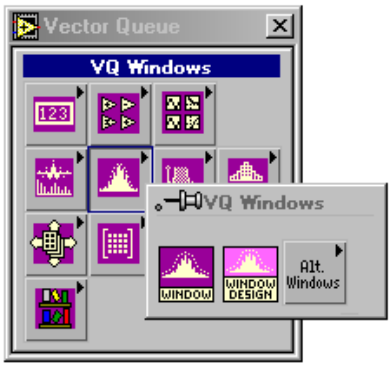
5.4.12 VQ Integral x(t).

VQ_X{n} = VQ input.

VQ_Integral X{n} = VQ output.

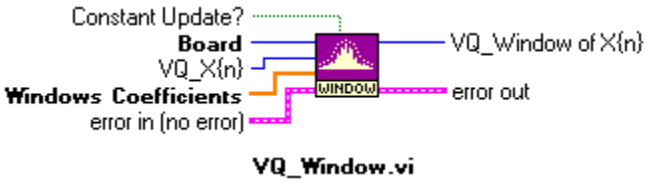
NOTE: *It is possible to observe a wrap around effect (also known as wind up) when using the Integration function. This is not an error in the DSP code, but rather the compounded effect of the input signal's offset added back onto itself. The Calibrate function within the miscellaneous operations subset may be used to digitally correct this initial input signal offset.*

5.5 VQ Windows.



5.5.1 VQ_Window.

The VQ_Window.vi function applies a windowing function to any input vector; may be used in conjunction with the Windows Coefficients Design.vi.



Constant Update? = LabVIEW boolean input that determines if the window coefficients are to be constantly updated or not. Defaulted to not constantly update in order to maximize throughput.

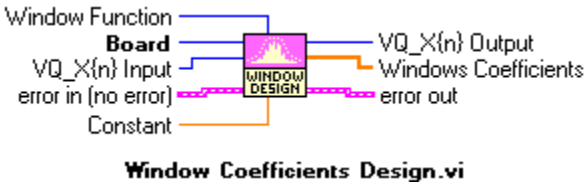
VQ_X{n} = VQ input, data to be subjected to a windowing function.

Windows Coefficients = LabVIEW input array of the coefficients that represent the windowing function of interest; may be derived from the Windows Coefficients Design.vi.

VQ_Window of X{n} = VQ output, windowed function.

5.5.2 Window Coefficients Design.

The Windows Coefficients Design.vi computes the coefficients for a specific windowing function in the LabVIEW/host domain, very convenient to be used used in conjunction with the VQ_Window.vi function.



Window Function = LabVIEW input to select the algorithm with which to compute the windowing coefficients of interest.

VQ_X{n} Input = VQ input, only used to derive the size of the vector since it

must coincide with the number of windowing coefficients.

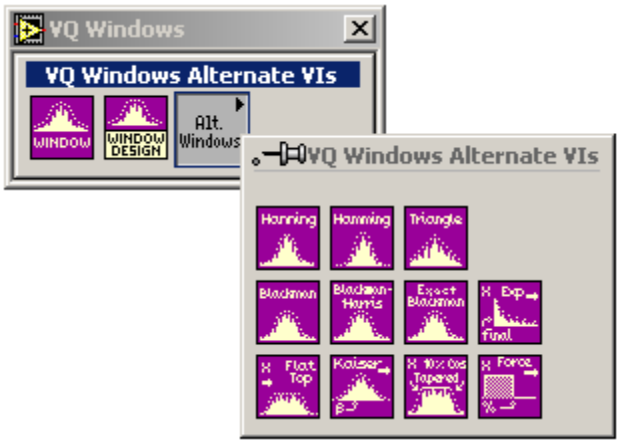
Constant = LabVIEW input, ancillary variable necessary to compute coefficients for some variety of windowing functions.

VQ_X{n} Output = VQ output that only mirrors the VQ_Input terminal as a convenience, may be optionally used to feed the VQ input terminal to the VQ_Window.vi function.

Windows Coefficients = LabVIEW output array, actual windowing coefficients to be fed into the same terminal on the VQ_Window.vi function.

5.5.3 VQ Windows Alternate VIs.

These functions are redundant as they duplicate the functionality of the Window Coefficients Design.vi and the VQ_Window.vi functions, and are not necessary. Only provided as a convenience if only one function icon is desired to represent a single windowing algorithm.



5.5.3.1 VQ_Hanning.

VQ_X{n} = VQ input.

VQ_Hanning of X{n} = VQ output.

5.5.3.2 VQ_Hamming.

VQ_X{n} = VQ input.

VQ_Hamming of X{n} = VQ output.

5.5.3.3 VQ_Triangle.

VQ_X{n} = VQ input.

VQ_Triangle of X{n} = VQ output.

5.5.3.4 VQ_Blackman.

VQ_X{n} = VQ input.

VQ_Blackman of X{n} = VQ output.

5.5.3.5 VQ_Exact Blackman.

VQ_X{n} = VQ input.

VQ_Exact Blackman of X{n} = VQ output.

5.5.3.6 VQ_Blackman Harris.

VQ_X{n} = VQ input.

VQ_Blackman Harris of X{n} = VQ output.

5.5.3.7 VQ_Flat Top.

VQ_X{n} = VQ input.

VQ_Flat Top of X{n} = VQ output.

5.5.3.8 VQ_Kaiser.

VQ_X{n} = VQ input.
Beta = LabVIEW input that defines the Beta weighting variable.

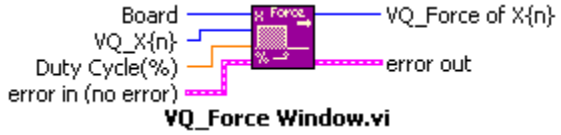
VQ_Kaiser of X{n} = VQ output.

5.5.3.9 VQ_Cosine Tapered.

VQ_X{n} = VQ input.

VQ_Cosine Tapered of X{n} = VQ output.

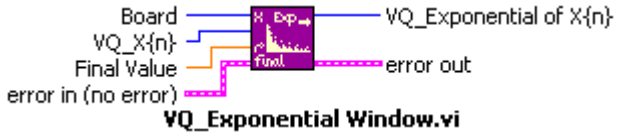
5.5.3.10 VQ_Force.



VQ_X{n} = VQ input.
Duty Cycle(%) = LabVIEW input to determine the duty cycle.

VQ_Force of X{n} = VQ output.

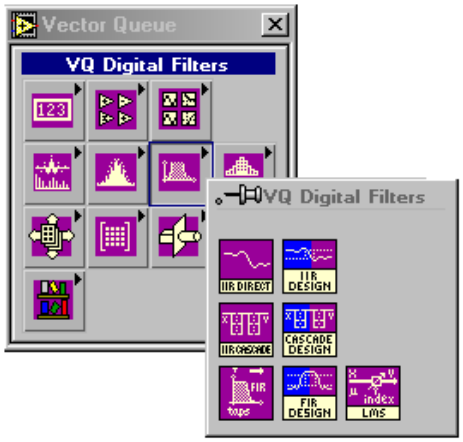
5.5.3.11 VQ_Exponential.



VQ_X{n} = VQ input.
Final Value = LabVIEW input to determine final output value of the input signal

VQ_Exponential of X{n} = VQ output.

5.6 VQ Digital Filters.



5.6.1 VQ_IIR Filter.

The IIR filters are setup as 2 separate VIs: a conventional LabVIEW based filter design VI and a DSP VQ_IIR Coefficient Filter Design VI. The VQ_IIR Filter VI is implemented on the DSP using the well known summation of two terms or recursive formula. Please refer to the Analysis Library Reference Manual.

Constant Update Coefficients? = LabVIEW boolean input that determines if the filter coefficients are to be constantly updated or not. Defaulted to not constantly update in order to maximize throughput.

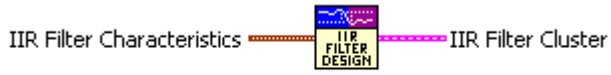
VQ_X{n} = VQ input.

IIR Filter Cluster = LabVIEW cluster containing the forward and reverse coefficients. The cluster may be derived by using one of the outputs of the LabVIEW

based design VIs.

VQ_Filtered X{n} = VQ output that returns a filtered vector only on the input vector in question.

5.6.1.1 IIR Direct Filter Coefficients Design.



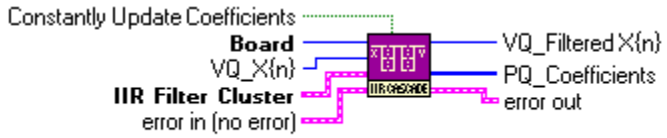
IIR Direct Filter Coefficients Design.vi

IIR Filter Characteristics = LabVIEW input cluster to define the IIR filter characteristics.

IIR Filter Cluster = LabVIEW cluster containing the forward and reverse coefficients.

5.6.2 VQ_IIR Cascade Filter.

The IIR filters are setup as 2 separate VIs: a conventional LabVIEW based filter design VI and a DSP VQ_IIR Cascade Coefficient Design VI. The VQ_IIR Cascade Filter VI is implemented on the DSP using the well known cascade formula. Please refer to the Analysis Library Reference Manual.



VQ_IIR Cascade Filter.vi

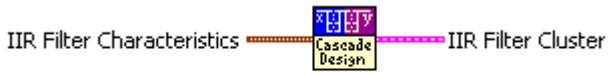
Constant Update Coefficients? = LabVIEW boolean input that determines if the filter coefficients are to be constantly updated or not. Defaulted to not constantly update in order to maximize throughput.

VQ_X{n} = VQ input.

IIR Filter Cluster = LabVIEW cluster containing the forward and reverse coefficients. The cluster may be derived by using one of the outputs of the LabVIEW based design VIs.

VQ_Filtered X{n} = VQ output that returns a filtered vector only on the input vector in question.

5.6.2.1 IIR Cascade Filter Coefficients Design.



IIR Cascade Filter Coefficients Design.vi

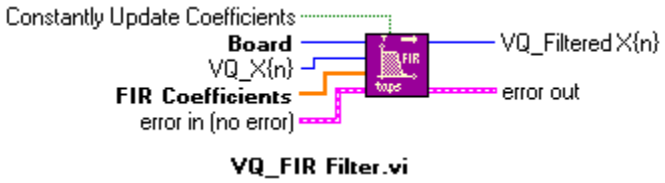
IIR Filter Characteristics = LabVIEW input cluster to define the IIR filter characteristics.

IIR Filter Cluster = LabVIEW cluster containing the forward and reverse coefficients.

5.6.3 VQ_FIR Filter.

The FIR filters are setup as 2 separate VIs: a conventional LabVIEW based filter design VI and a VQ_FIR Filter VI. The VQ_FIR Filter VI is implemented on the DSP using the well known finite convolution or recursive formula. Please refer to the Analysis Library Reference Manual.

The LabVIEW based VQ_FIR Filter Design VI is merely an example of a way to design and build an array of FIR coefficients. An array of FIR coefficients can be built alternatively by using the FIR filter VIs supplied with the LabVIEW Analysis Library. However, the diagram is much cleaner when used with the supplied single icon VQ_FIR Filter Design VI.



Constant Update Coefficients? = LabVIEW boolean input that determines if the filter coefficients are to be constantly updated or not. Defaulted to not constantly update in order to maximize throughput.

VQ_X{n} = VQ input.

FIR Coefficients = LabVIEW array containing the FIR coefficients. The array may be derived by using one of the outputs of the LabVIEW based design VIs.

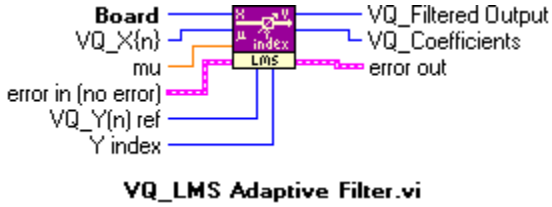
VQ_Filtered X{n} = VQ output that returns a filtered vector only on the input vector in question.

5.6.3.1 FIR Filter Coefficients Design.

FIR Filter Characteristics = LabVIEW input cluster to define the FIR filter characteristics.

FIR Filter Cluster = LabVIEW output cluster containing the forward coefficients.

5.6.4 VQ LMS Adaptive Filter.



VQ_X{n} = VQ input.

Scale Factor (mu) = LabVIEW constant, representing the compensation factor that is multiplied by the error between the actual signal and the desired signal.

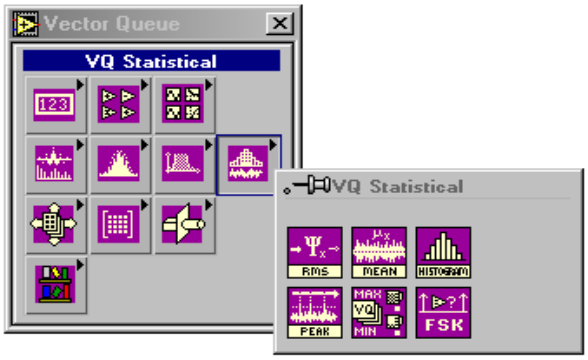
VQ_Y{n} ref = VQ input, reference source signal.

Y Index = LabVIEW input, determines the index point of the input Y[n} vector.

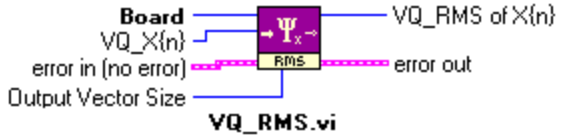
VQ_Filtered Output = VQ output that returns a continuous filtered value without the need for initial conditions, analogous to a conventional analog filter.

VQ_Coefficients = VQ output that simply mirrors in the VQ domain the actual coeficients used by the LMS algorithm.

5.7 VQ Statistical.



5.7.1 VQ RMS.

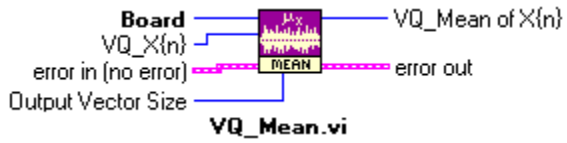


VQ_X{n} = VQ input.

Output Size = LabVIEW input. Specifies the output vector size such that each element is the RMS of a Vector.

VQ_RMS of X{n} = VQ output.

5.7.2 VQ_Mean.



VQ_X{n} = VQ input.
Output Vector Size = LabVIEW input. Specifies the output vector size such that each element is the Mean of a Vector.

VQ_Mean of X{n} = VQ output.

5.7.3 VQ_Histogram.

This VI will produce an output array that contains a histogram of the input array. The Range input determines what range of values will be included in the histogram. The lowest value that the Histogram will use is 0. This means that any zero centered data must be offset so that it is centered in the specified range.

The Histogram function places data in bins only on integer boundaries. Therefore, if higher precision is needed, the data should be scaled appropriately.

EXAMPLE: If the data to be used ranges from -5 to +5, and we wish to see intervals of 0.5 the data would have to be scaled as follows:

(input array + 5) * 2

Adding 5 centers our data in the array, and multiplying by two allows the 0.5 intervals. The RANGE input for this example would need to be set to 20.

VQ_X{n} = VQ input.
Range = LabVIEW input, range of values which Histogram will be included in the histogram.

VQ_Histogram = VQ output.

5.7.4 VQ_Threshold Peak Detect.

This VI outputs an array containing the indexes where peaks were located. A Peak count

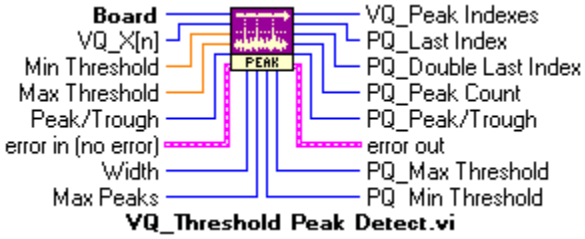
is also provided.

Peak/Trough allows the user to select whether the VI will detect positive going peaks, or negative going troughs. A 0 selection indicates peaks, a 1 selection indicates troughs.

A peak is determined when the input signal exceeds the min threshold, then goes below it again, without exceeding the max threshold. If the max threshold is exceeded, the peak is ignored. Width indicates the minimum number of points above the min threshold in order for the peak to be counted. If this parameter is set to 0 then all peaks will be detected regardless of width.

Last index indicates the index of the last peak in this array.
Double last index indicates the index of the last peak in the previous array. (NOTE: this index will always be negative since the first point in the current array is at index 0)

Max peaks indicates the maximum number of peaks that the function will count. If this number is exceeded, subsequent peaks are ignored.

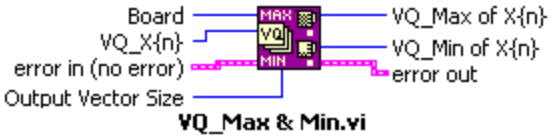


VQ_X{n} = VQ input to be evaluated
Min Threshold = LabVIEW input, minimum value to consider it is a peak
Max Threshold = LabVIEW input, maximum value to consider it is a peak.
Peak/Trough = Detect Peak or Trough
0 = *Peak*
1 = *Trough*
Width = Number of points to be sampled
Max Peaks = Maximum number of peaks to count / detect

VQ_Peak Indexes = VQ output, Index numbers, indicating where the peaks/troughs occurred.
PQ_Past Index = PQ output, index of the last peak.
PQ_Double Last Index = PQ output.
PQ_Peak Count = PQ output, count of peaks/troughs detected.
PQ_Peak/Trough = PQ output, highest Peak/Trough value detected in the vector.
PQ_Max Threshold = PQ output, value of Maximum Threshold
PQ_Min Threshold = PQ output, value of Minimum Threshold

5.7.5 VQ_Max & Min.

Obtains the maximum and minimum values of a VQ vector.



VQ_X{n} = VQ input.

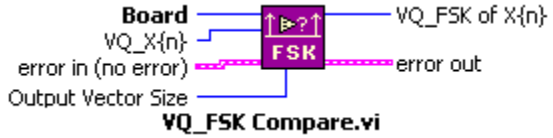
Output Vector Size = LabVIEW input. Specifies the output vector size such that each element is the Max or Min of a Vector.

VQ_Max of X{n} = VQ output, vector with the maximum values detected in a vector.

VQ_Min of X{n} = VQ output, vector with the minimum values detected in a vector.

5.7.6 VQ_FSK Compare.

Frequency Shift Keyed ? ? ?

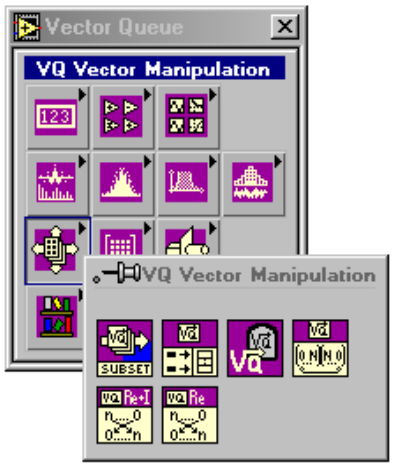


VQ_X{n} = VQ input ? ? ?

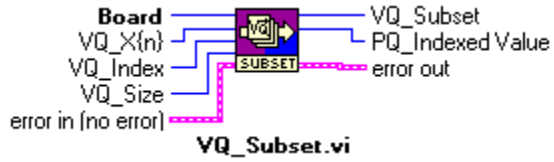
OutputVector Size = LabVIEW input, defines the size of VQ_FSK of X{n}? ? ?

VQ_FSK of X{n} = VQ output ? ? ?

5.8 VQ Vector Manipulation.



5.8.1 VQ_Subset.

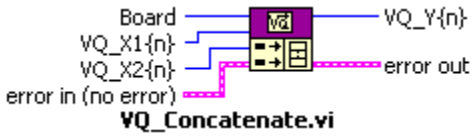


VQ_X{n} = VQ input.
VQ_Index = VQ input, used to determine the location of the first point within the current vector that starts the new vector block.
VQ_Size = VQ input, used to determine the size of the new vector block.

VQ_Subset = VQ output, new vector subset.
PQ_Indexed Value = PQ output, value extracted from the vector as indexed by the 'VQ_Index' input.

5.8.2 VQ_Concatenate Vectors.

Concatenates the two input vectors into a single output vector



VQ_X1{n} = VQ input vector
VQ_X2{n} = VQ input vector

VQ_Y{n} = Resulting single VQ output, with VQ_X2{n} concatenated after VQ_X1{n}.

5.8.3 VQ_Mirror.

This VI is useful for turning the output of a single sided Real FFT into a full sided Real FFT.

VQ_X{n} = VQ input.

VQ_Y{n} = VQ output that is doubled in size, with the new half having its elements reversed.

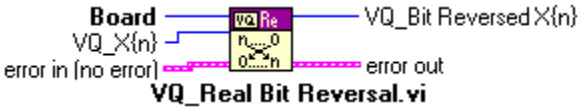
5.8.4 VQ_Reverse.

VQ_X{n} = VQ input.

VQ_Y{n} = VQ output with its elements reversed.

5.8.5 VQ_Real Bit Reversal.

Performs a bit reversal function on the real part only, used as a subVI for FFTs.

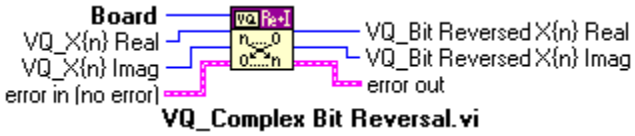


VQ_X{n} = VQ input.

VQ_Bit Reversed X{n} = VQ output.

5.8.6 VQ_Complex Bit Reversal.

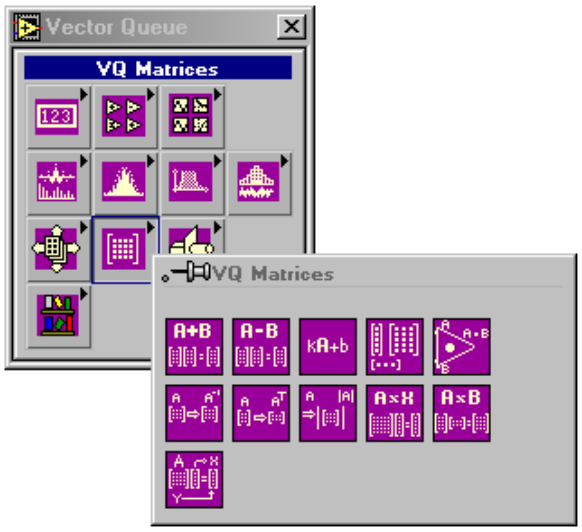
Performs a bit reversal function on both the real and imaginary parts, used as a subVI for Complex FFTs.



VQ_X{n} Real = VQ input.
VQ_X{n} Imag = VQ input.

VQ_Bit Reversed X{n} Real = VQ output.
VQ_Bit Reversed X{n} Imag = VQ output.

5.9 VQ Matrices.



5.9.1 VQ Matrix Addition.

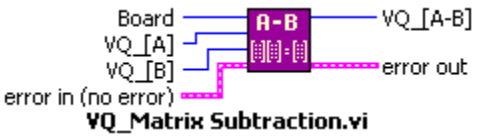
Computes the addition of two NxM matrices.

VQ_[A] = VQ Input vector
VQ_[B] = VQ Input vector

VQ_[A+B] = VQ output, elements are the addition of each corresponding A and B element.

5.9.2 VQ Matrix Subtraction.

Computes the subtraction of one NxM matrix from another.

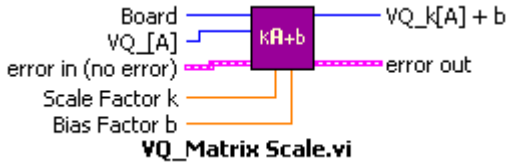


VQ_[A] = VQ Input vector
VQ_[B] = VQ Input vector

VQ_[A-B] = VQ output, elements are the difference of each corresponding A and B element.

5.9.3 VQ Matrix Scale.

Scales each element of an NxM matrix with a gain factor "k" and a bias factor "b."



VQ_[A] = VQ Input vector
Scale Factor k = LabVIEW input, Gain factor of scale
Scale Factor b = LabVIEW input, Bias factor of scale

VQ_k[A] + b = VQ output which elements are obtained multiplying each element of VQ_A by the gain factor "k", then added the bias factor "b"

5.9.4 VQ_Matrix Outer Product.

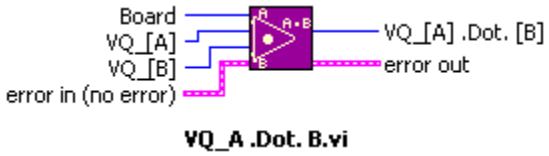
Computes the Outer Product of two NxM matrices.

VQ_[A] = VQ input vector
VQ_[B] = VQ input vector

VQ_[Ai][Bj] = VQ Outer Product of VQ_A by VQ_B

5.9.5 VQ_A .Dot. B.

Computes the "Scalar" or "Dot" product of two Nx1 matrices.

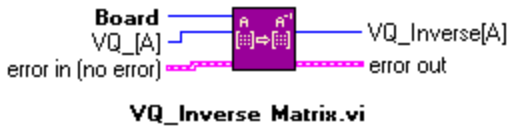


VQ_[A] = VQ input vector
VQ_[B] = VQ input vector

VQ_[A].Dot.[B] = VQ Dot Product of VQ_A by VQ_B

5.9.6 VQ_Inverse Matrix.

Computes the Inverse of an NxM matrix.

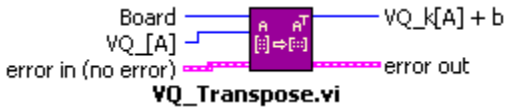


VQ_[A] = VQ input matrix.

VQ_Inverse[A] = The resultant Inverse of VQ_[A] matrix

5.9.7 VQ_Transpose.

Transposes an NxM matrix.



VQ_[A] = VQ input matrix.

VQ_Transpose[A] = The resultant Transposed of VQ_[A] matrix

5.9.8 VQ_Determinant.

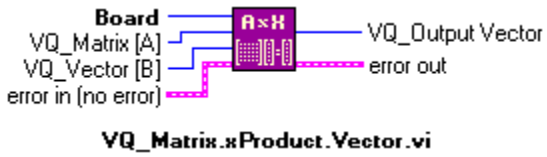
Computes the Determinant of a square NxN matrix.

VQ_[A] = VQ input matrix.

VQ_Determinant[A] = The resultant Determinant matrix of VQ_[A]

5.9.9 VQ_AxX{n}.

Computes the "Cross" product of an [NxR] matrix times an [Rx1] matrix or vector, and returns a single Nx1 matrix or vector.

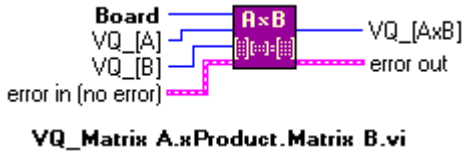


VQ_[A] = VQ input vectors
VQ_X{n} = VQ input vectors

VQ_[A x X{n}] = VQ Cross Product of VQ_A by VQ_X{n}.

5.9.10 VQ_AxB.

Computes the "Cross" product of an [NxR] matrix times an [RxM] matrix, and returns a single NxM matrix.



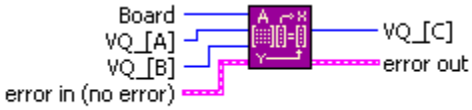
VQ_Matrix A.xProduct.Matrix B.vi

VQ_[A] = VQ input vectors
VQ_[B] = VQ input vectors

VQ_[A x B] = VQ Cross Product of VQ_A by VQ_B

5.9.11 VQ_Solve Linear Equations.

Solves for a linear system, with variables represented by a square NxN matrix.



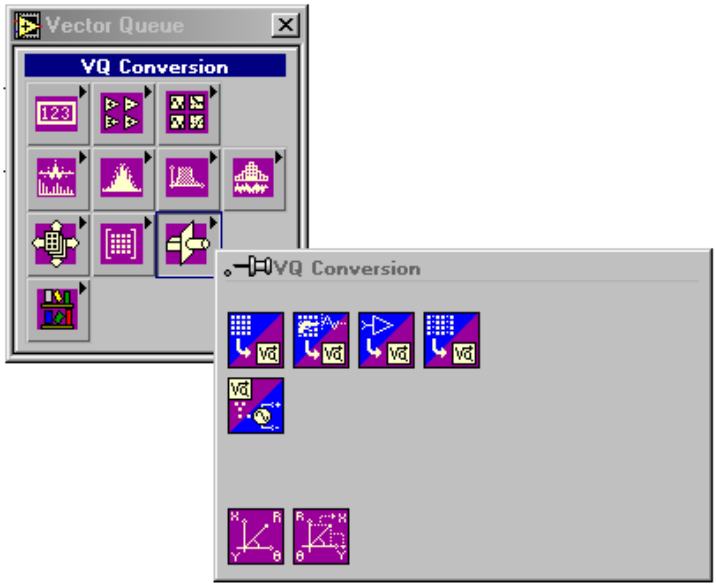
VQ_Solve Linear Equations .vi

VQ_[A] = VQ input matrix
VQ_[B] = VQ input matrix

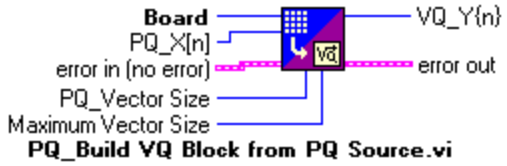
VQ_[C] = Resultant VQ matrix containing the linear parameters for the input matrices

---

5.10 VQ Conversion.



5.10.1 VQ Build VQ Block From PQ Source.



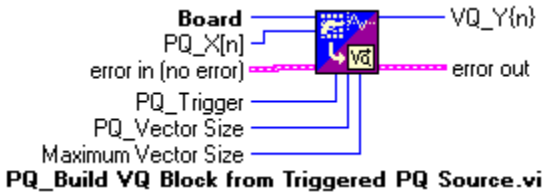
PQ_X[n] = PQ input used to determine the input source from which to build a vector block.

PQ Vector Size = PQ input, used to determine the number of points from which to build the vector block.

Maximum Vector Size = LabVIEW input, used to set a limit for a maximum vector size. Useful for reserving a maximum fixed VQ buffer, thereby allowing the dynamic resize of the resultant vector within the DSP domain.

VQ_Y{n} = VQ output, containing the number of points as defined by the 'PQ_Vector Size' input.

5.10.2 VQ Build VQ Block From Triggered PQ Source.



PQ_X[n] = PQ input used to determine the triggered input source from which to build a vector block.

PQ_Trigger = PQ input, used as an enable or trigger marker to begin collecting

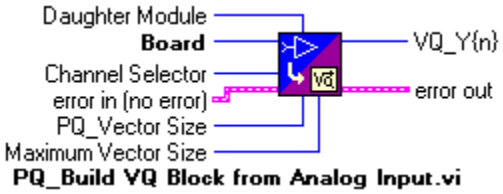
data to fill the vector.

PQ Vector Size = PQ input, used to determine the number of points from which to build the vector block.

Maximum Vector Size = LabVIEW input, used to set a limit for a maximum vector size. Useful for reserving a maximum fixed VQ buffer, thereby allowing the dynamic resize of the resultant vector within the DSP domain.

VQ_Y{n} = VQ output, containing the number of points as defined by the 'PQ_Vector Size' input.

5.10.3 VQ Build VQ Block From Analog Input.



Channel Selector = LabVIEW input used to select the analog input channel of interest.

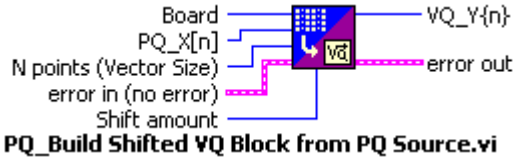
PQ Vector Size = PQ input, used to determine the number of points from which to build the vector block.

Maximum Vector Size = LabVIEW input, used to set a limit for a maximum vector size. Useful for reserving a maximum fixed VQ buffer, thereby allowing the dynamic resize of the resultant vector within the DSP domain.

VQ_Y{n} = VQ output, containing the number of points as defined by the 'PQ_Vector Size' input.

5.10.4 VQ Build Shifted VQ Block from PQ Source.

This VI will build a block much like the VQ_Build VQ Block From PQ Source VI. The difference is that the VI maintains the previous N-1 points, and adds the new point on to the end of the array. This provides a constantly shifting array that is useful for convolutions and filtering applications.



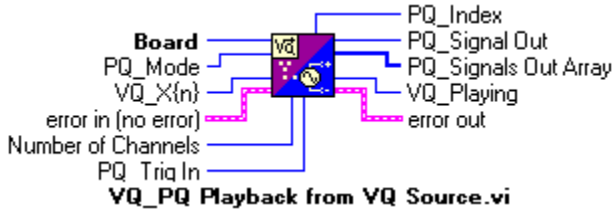
PQ_X[n] = PQ values.

N points (Vector Size) = LabVIEW input, used to determine the number of points to group into a VQ vector.

Shift amount = LabVIEW input, the amount with which to shift the incoming PQ data set.

VQ_Y{n} = VQ vector output

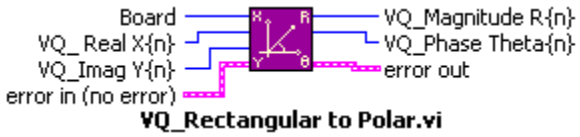
5.10.5 PQ Playback from VQ Source.



VQ_X{n} = VQ input, in a vector (array) form.
PQ_Mode = PQ input that selects between single shot or continous signal generation.
0 = Continous signal generation; default.
nonzero = Single shot generation.
PQ_Trig In = PQ input that enables or disables signal generation.
0 = Consantly enables signal generation; default.
nonzero = External mode control, disables signal generation.
Number of Channels = LabVIEW input to determine the number of output channels to reproduce the PQ output.

PQ_Index = PQ output of the index number indicating the relative position of the current output point within the waveform buffer; starts at '0' and ends at 'n-1'.
PQ_Signal Out = PQ output of signal being played back (point form).
PQ_Signals Out Array = PQ output array of multiple signals being played back (point form), number of signals played back depends on the 'Number of Channels' variable.
VQ_Playing = VQ output of buffer/signal being played back (vector form); used by PQ_Waveform Playback VI.

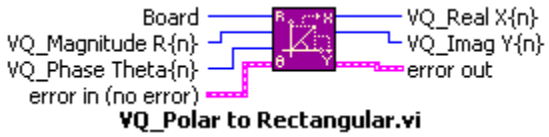
5.10.6 VQ_Rectangular to Polar.



VQ_Real X{n} = VQ input.
VQ_Imag Y{n} = VQ input.

VQ_Magnitude R{n} = VQ output.
VQ_Phase Theta{n} = VQ output.

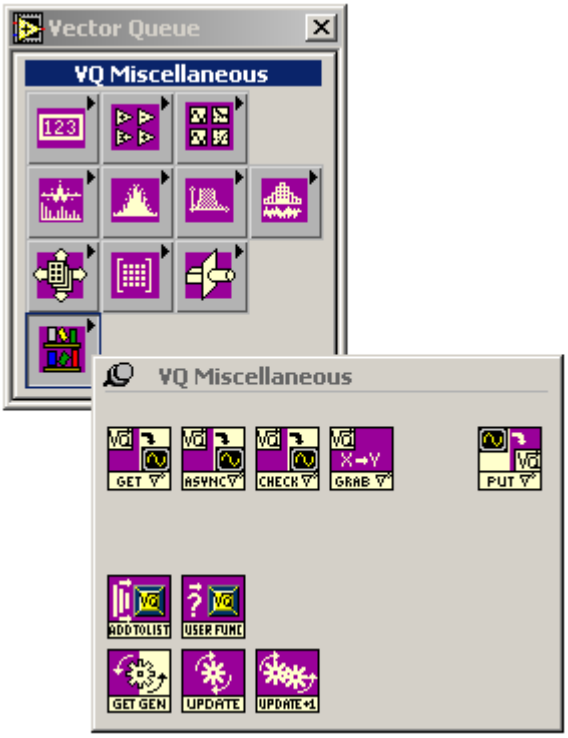
5.10.7 VQ_Polar to Rectangular.



VQ_Magnitude R{n} = VQ input.
VQ_Phase Theta{n} = VQ input.

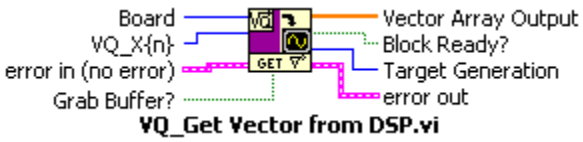
VQ_Real X{n} = VQ output.
VQ_Imag Y{n} = VQ output.

5.11 VQ Miscellaneous.



5.11.1 VQ Get Vector from DSP.

This VI, when running in Wait mode, assumes the host will be fast enough to pickup every processed vector. This assumption requires a very fast host or a slow set of vector operations. The False case will pick up a buffer regardless of whether it is ready. The Block ready output displays if this has occurred.



VQ_X{n} = VQ input designating the address where the incoming vector is to be placed.

Grab Buffer? = LabVIEW boolean input, used to indicate whether or not to accept an upcoming vector input.

F = Grab: When False, the currently available input vector is used.

T = Wait: Forces the processing to wait for a valid synchronized input vector, useful in applications where no points can be dropped.

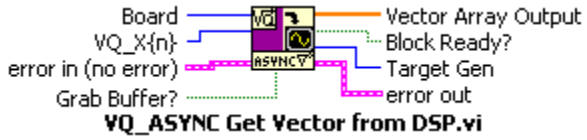
Vector Array Output = LabVIEW array of binary data read from DSP.

Block Ready? = LabVIEW boolean that indicates whether or not the current VQ input is available or not.

Target Generation = LabVIEW output that indicates the current generation number for the current VQ input; used mainly for debugging.

5.11.2 VQ_ASYNC Get Vector from DSP.

This VI is similar to Get Vector from DSP. It is a littler slower but allows the host to pickup data when the host will lag behind the DSP. The Grab/Wait and ready terminals perform as mentioned in Get Vector from DSP.



VQ_X{n} = VQ input designating the address where the incoming vector is to be placed.

Grab Buffer? = LabVIEW boolean input, used to indicate whether or not to accept an upcoming vector input.

F = Grab: When False, the currently available input vector is used.

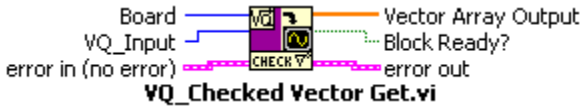
T = Wait: Forces the processing to wait for a valid synchronized input vector, useful in applications where no points can be dropped.

Vector Array Output = LabVIEW array of binary data read from DSP.

Block Ready? = LabVIEW boolean that indicates whether or not the current VQ input is available or not.

Target Generation = LabVIEW output that indicates the current generation number for the current VQ input; used mainly for debugging.

5.11.3 VQ_Checked Vector Get.



VQ_Input = VQ input.

Vector Array Output = LabVIEW array of binary data read from DSP.
Block Ready? = LabVIEW boolean that indicates whether or not the current VQ input is available or not.

5.11.4 VQ Grab Vector.

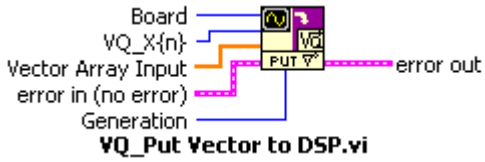
This function just copies the VQ input vector and places it into the output vector. Only copies only when generation is zero.

VQ_X{n} = VQ input.

VQ_Y{n} = VQ output.

5.11.5 VQ Put Vector to DSP.

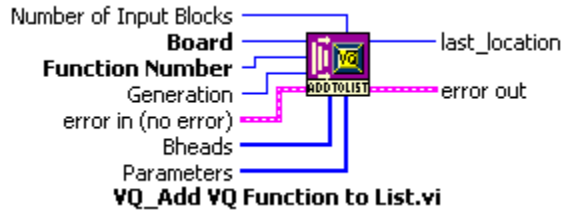
This VI performs a similar function as VQ_Constant, except that this one doesn't allocate memory space in the DSP; instead, it just writes to an allocated VQ block.



VQ_X{n} = VQ input.
Vector Array Input = LabVIEW array of binary data to be sent to the DSP.
Generation = LabVIEW input.

5.11.6 VQ Add VQ Function to List.

Adds a VQ Function to VQ List, inside DSP board.



Number of Input Blocks = Number of input VQ vector blocks
Function Number = A number which identifies the funcion, with what DSP knows what to do
Generation = An incremental value to know when data is updated and ready to

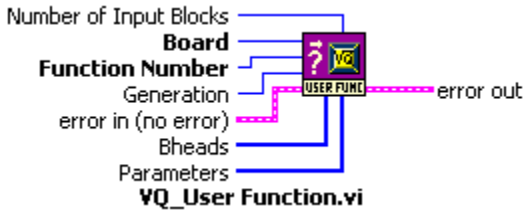
be processed.

Bheads = An array of pointers to the several vectors needed for current function
Parameters = An array of parameters or constant values needed by the function

last_location = Last DSP address, generally used by output of function.

5.11.7 VQ User Function.

Available for custormer to add custom VQ Functions



Number of Input Blocks = Number of input VQ vector blocks
Function Number = A number which identifies the funcion, with what DSP knows what to do
Generation = An incremental value to know when data is updated and ready to be processed.
Bheads = An array of pointers to the several vectors needed for current function
Parameters = An array of parameters or constant values needed by the function

5.11.8 VQ Get Generation.

Gets Current Generation number

Generation = Current generation

5.11.9 VQ Update Generation.

Updates Generation of a VQ vector, with generation value of another VQ. Not intneded for general use.

VQ_X{org} = VQ vector, from where generation must be taken from.
VQ_Y{copy} = VQ vector, where generation must be copied/updated to.

VQ_Y{out} = same as VQ_Y{copy}, except after execution, generation of vector is same as of VQ_X{org}..

5.11.10 VQ_Update +1 Generation.

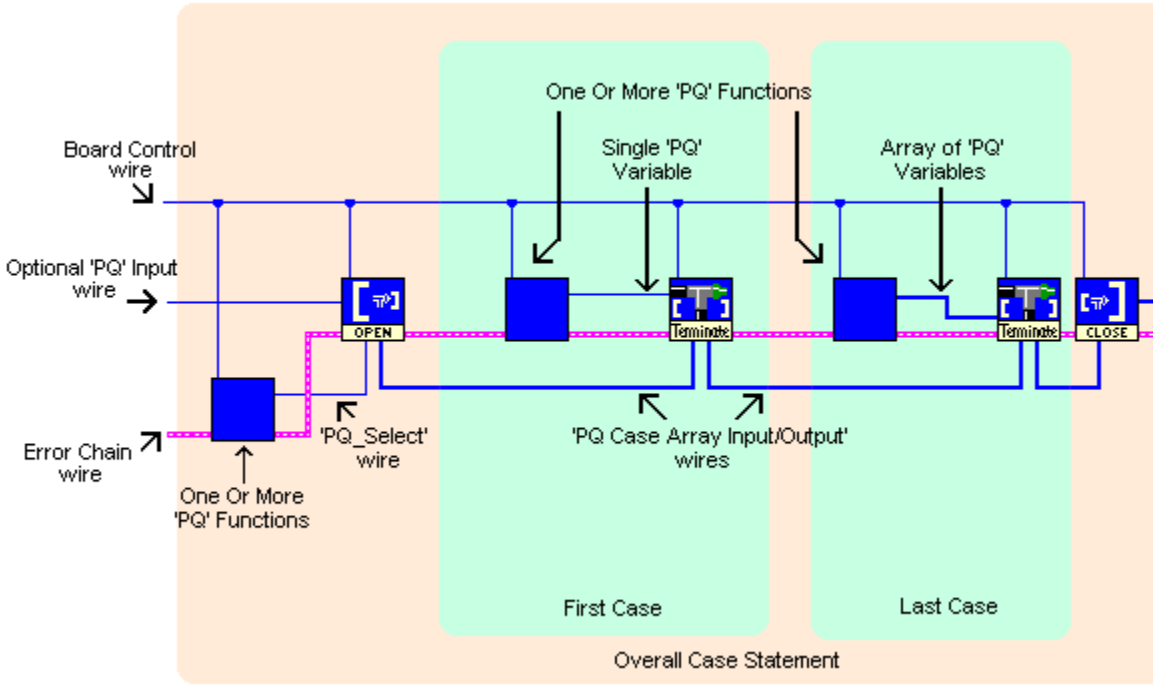
Increments a Generation numer

VQ_X{orig} = VQ vector, from where generation must be taken from.
VQ_Y{copy} = VQ vector, where generation must be copied/updated to.

VQ_Y{out} = same as VQ_Y{copy}, except after execution, generation of vector is also incremented by 1.

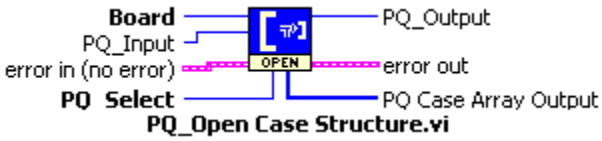
**$$$

Below is a diagram illustrating the general form that one must follow in order to implement case structures for PQ functions.



6.1.1 PQ_Open Case Structure.

Beginning of a PQ Case Structure.



PQ_Input = PQ input, optionally used to force precedence.
PQ_Select = PQ value for selecting the case to be executed, ranges from 0 to N-1 where N is the number of cases. If case selection is invalid, none of the cases are executed and the results of the previous execution are preserved.

PQ_Output = PQ output, optionally used to force precedence.
PQ Case Array Output = LabVIEW output array, must be connected to the next PQ_Case.vi

6.1.2 PQ_Terminate This Case.

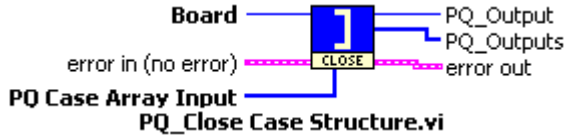
End of each case within a PQ Case Structure.

PQ_Input = Queuing Input to pass results from the case.
PQ_Input Array = PQ input source array, if wired,data from this array is used and PQ_Input is ignored.
PQ Case Array Input = PQ_Input connects from the PQ Case Array Output terminal of the previous PQ_Case.vi.

PQ Case Array Output = LAbVIEW output array, connects to the PQ Case Array Input terminal of the next PQ_Case.vi

6.1.3 PQ_Close Case Structure.

End of PQ Case Structure.

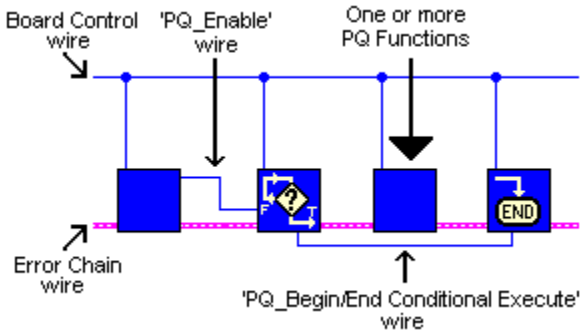


PQ Case Array Input = PQ_Input connects from the PQ Case Array Output terminal of the previous PQ_Case.vi.

PQ_Output = Output PQ value.
PQ_Outputs = PQ output Array.

6.2 PQ Conditional Execute Structures.

Below is a diagram illustrating the general form that one must follow in order to implement conditional execute structures for PQ functions.



6.2.1 PQ_Begin Conditional Execute.

Executes functions depending on the value of PQ_Enable, if the value of PQ_Enable

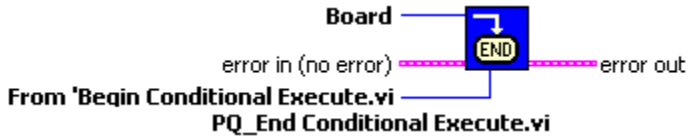
input is 1, this VI will allow execution of subsequent functions upto "PQ End Conditional Execute" during the current iteration, else if the value
is 0, none of these functions are executed.

PQ_Enable (Execute) = PQ input, with possible values of zero or else.

To "End Conditional Execute" = LabVIEW output, values to be passed to "From Begin Conditional Execute" terminal of PQ_End Conditional Execute.vi

6.2.2 PQ End Conditional Execute.

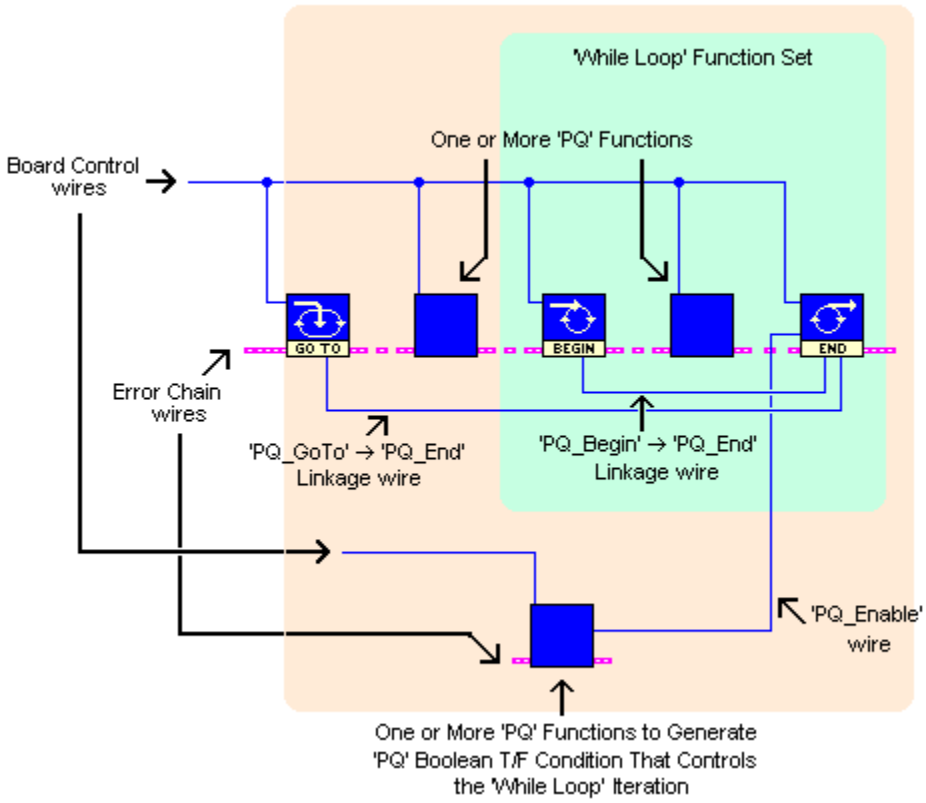
Ends the conditional execution of functions.



From "Begin Conditional Execute.vi" = LabVIEW input, from "To PQ End Conditional Execute" terminal of PQ_Begin Conditional Execute.vi.

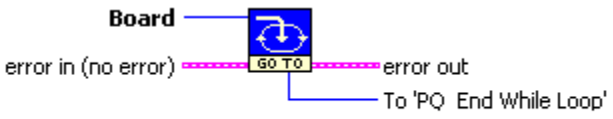
6.3 PQ While Loop Structures.

Below is a diagram illustrating the general form that one must follow in order to implement while loop structures for PQ functions.


'While Loop' Function Set
One or More 'PQ' Functions
Board Control wires
GO TO
BEGIN
END
Error Chain wires
'PQ_GoTo' → 'PQ_End' Linkage wire
'PQ_Begin' → 'PQ_End' Linkage wire
'PQ_Enable' wire
One or More 'PQ' Functions to Generate 'PQ' Boolean T/F Condition That Controls the 'While Loop' Iteration

6.3.1 PQ_GoTo While Loop Structure.

An ancillary function required as a marker for the PQ While Loop Structure, but itself is not involved nor is subject to the loop condition. It must be the first PQ function in the PQ list when a PQ While Loop is desired. Although this vi can only be used once and must also be the first PQ vi in the data flow, there can be as many individual as well as nested loops as required in the aplication.



PQ_GoTo While Loop Structure.vi

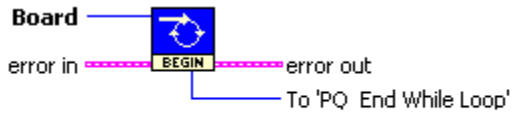
To "PQ End While Loop" = LabVIEW output, must be connected to all 'PQ_End While Loop Structure' VIs in the application.

6.3.2 PQ_Begin While Loop Structure.

Beginning of a group of PQ VIs to be executed when PQ_End While Loop Structure receives a PQ True condition.

***NOTE**: It is very important that the VI which evaluates the TRUE/FALSE condition be included between this vi and PQ_End While Loop Structure; otherwise, once the*

condition is evaluated to be True, the vi cannot execute to change the condition, and the loop becomes infinite.

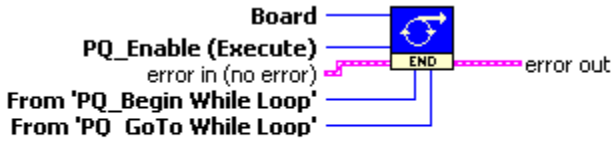


PQ_Begin While Loop Structure.vi

To "PQ End While Loop" = LabVIEW output, which connects to "From PQ_Begin While Loop" terminal of the PQ_End While loop structure.vi .

6.3.3 PQ End While Loop Structure.

End of a PQ While Loop Structure. When the PQ_Enable input is True, the while loop is executed between PQ_Begin While Loop Structure.vi and this vi. Thus, it is very important that the VI which evaluates the TRUE / FALSE condition be included in the while loop; otherwise, once the condition is evaluated to be True, the vi cannot execute to change the condition, and the loop becomes infinite.



PQ_End While Loop Structure.vi

PQ_Enable (Execute) = PQ input, when True, the while loop is executed between PQ_Begin While Loop Structure.vi and this vi. As a rule, the PQ vi connected to this terminal must be inside the loop, as well as all the other VI's which are needed for the evaluation.

From "PQ_Begin While Loop" = LabVIEW input, must be connected to "PQ_End While Loop Structure" terminal of the PQ_Begin While Loop.vi.

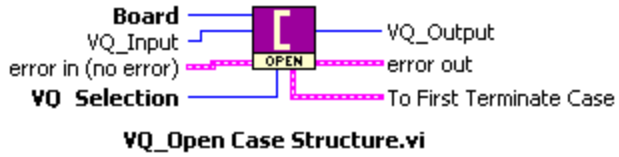
From "PQ_GoTo While Loop" = LabVIEW input, must be connected to "PQ_End While Loop Structure" terminal of the PQ_GoTo While Loop.vi

---

6.4 VQ Case Structures.

6.4.1 VQ Open Case Structure.

Beginning of a VQ Case Structure.



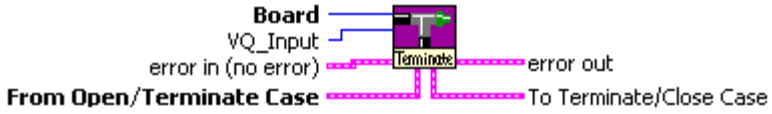
VQ_Open Case Structure.vi

VQ_Input = Queuing input for VQ variables optionally used to force precedence.
VQ Selection = VQ value for selecting the case to be executed, ranges from 0 to N-1 where N is the number of cases. If case selection is invalid, none of the cases are executed and the results of the previous execution are preserved.

VQ_Output = VQ output, optionally used to force precedence.
To First Terminate Case = LabVIEW cluster output, must be connected to the "From Open/Terminate Case " terminal of the next VQ Terminate This Case.vi.

6.4.2 VQ Terminate This Case.

End of each case within a VQ Case Structure



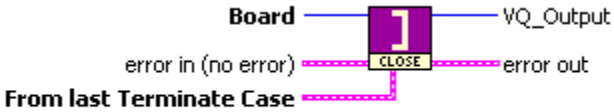
VQ_Terminate This Case.vi

VQ_Input = VQ input, to pass results from the case.
From Open/Terminate Case = LabVIEW cluster input, must be connected to the "To First Terminate Case" terminal of the VQ Open Case.vi or "To Terminate/Close Case" terminal of the previous VQ Terminate Case.vi.

To Open/Terminate Case = LabVIEW cluster output, must be connected to the "From Open/Terminate Case " terminal of the next VQ Terninate This case.vi or to the "From Last Terminate case" terminal of the VQ Close Case.vi.

6.4.3 VQ Close Case Structure.

End of VQ Case Structure.



VQ_Close Case Structure.vi

From last Terminate Case = LabVIEW cluster input, must be connected to the "To Terminate/Close Case" terminal of the last VQ Terminate Case.vi.

VQ_Output = VQ output.

6.5 VQ Conditional Execute Structures.

6.5.1 VQ Begin Conditional Execute.

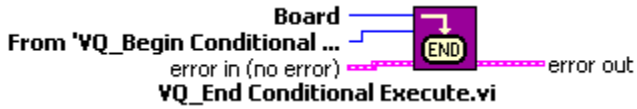
Executes functions depending on the value of VQ_Enable, if the value of VQ_Enable input is 1, this VI will allow execution of subsequent functions upto "VQ End Conditional Execute" , else if the value is 0, none of these functions are executed.

VQ_Enable (Execute) = VQ input, with possible values of zero or else.

To "VQ_End Conditional Execute" = LabVIEW output, value to be passed to "From VQ_Begin Conditional Execute" terminal of VQ_End Conditional Execute.vi

6.5.2 VQ End Conditional Execute.

Ends the conditional execution of functions.



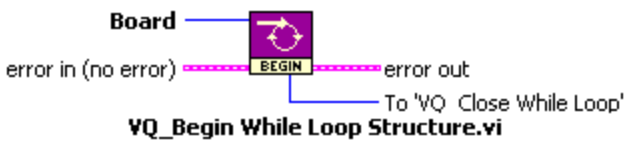
From "VQ_Begin Conditional Execute" = LabVIEW input, from "To VQ End Conditional Execute" terminal of VQ_Begin Conditional Execute.vi.

6.6 VQ While Loop Structures.

6.6.1 VQ Begin While Loop Structure.

Beginning of a group of VQ VIs to be executed when VQ_End While Loop Structure receives a VQ True condition.

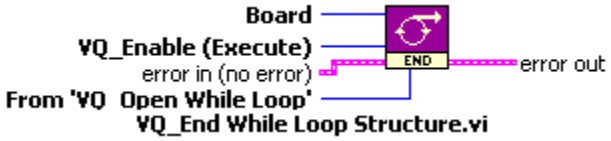
NOTE: *It is very important that the VI which evaluates the TRUE/FALSE condition be included between this vi and VQ_End While Loop Structure; otherwise, once the condition is evaluated to be True, the vi cannot execute to change the condition, and the loop becomes infinite.*



To "VQ Close While Loop" = LabVIEW output, must be connected to "From VQ_Begin While Loop" terminal of the VQ_End While loop structure.vi .

6.6.2 VQ_End While Loop Structure.

End of a VQ While Loop Structure. When the VQ_Enable input is True, the while loop is executed between VQ_Begin While Loop Structure.vi and this vi. Thus, it is very important that the VI which evaluates the TRUE / FALSE condition be included in the while loop; otherwise, once the condition is evaluated to be True, the vi cannot execute to change the condition, and the loop becomes infinite.



VQ_Enable (Execute) = VQ input, when True, the while loop is executed between VQ_Begin While Loop Structure.vi and this vi. As a rule, the VQ vi connected to this terminal must be inside the loop, as well as all the other VI's which are needed for the evaluation.

From "VQ_Open While Loop" = LabVIEW input, must be connected to "VQ_End While Loop Structure" terminal of the VQ_Begin While Loop.vi.

$$$

7.2 Boolean to Channel Array.

Converts an array of randomly selected boolean inputs to a numeric array whose values reflect the enabled boolean input's relative position in the array. Useful for translating boolean buttons that are used to select one or more input channels to a numeric array whose values reflect the actual channel numbers. Created because no native equivalent exists in the LabVIEW libraries.

Input Boolean Array = LabVIEW input of boolean array.

Output Channel Array = LabVIEW output of a numeric array.

7.3 Application Launch.

Launches a DOS/Windws application from within LabVIEW.

appName = LabVIEW input string, name of application to be launched.
appParameters = LabVIEW input string, parameters/switches needed to launch the application.
Default Folder = LabVIEW input string, folder where the application resides.
Window Mode = LabVIEW input, specifies if the mode of operation of the command line window after program execution.
Timeout (msec) = LabVIEW input, the time (in ms) to delay the entry into the next function after the program has executed.